Working Paper No. 338

AN INTRODUCTION TO STRUCTURED MODELING

by

ARTHUR M. GEOFFRION

Revised
February 1987

Contract N00014-75-C-0570

WESTERN MANAGEMENT SCIENCE INSTITUTE
University of California, Los Angeles

WESTERN MANAGEMENT SCIENCE INSTITUTE
University of California, Los Angeles

Working Paper No. 338

June, 1986
Revised February, 1987

AN INTRODUCTION TO STRUCTURED MODELING

by

Arthur M. Geoffrion

To appear in *Proceedings of the Conference on Integrated Modeling Systems* (held at the University of Texas, Austin, October 1986) and, without the section on implementation, in *Management Science*, May 1987.

I acknowledge with gratitude the substantial assistance and encouragement provided by many colleagues and students, including G. Bradley, S. Chari, R. Clemence, D. Dolk, C.K. Farn, J. Jackson, C. Jones, M. Lenard, J. Mamer, S. Maturana, Y. Tsai, and G. Wright.

My debt extends to the National Science Foundation and the Office of Naval Research for supporting this work since its inception, to the Naval Personnel R&D Center, and to Hewlett-Packard and IBM for their generous grants to the UCLA Graduate School of Management. The views contained in this report are mine and not to be attributed to the sponsoring agencies.

Abstract

The discipline of modeling has advanced only slowly compared to disciplines concerned with analyzing and solving models once they are brought into being. Structured Modeling is an attempt to redress this imbalance.

Structured Modeling aims to provide a formal mathematical *framework* and *computer-based environment* for conceiving, representing, and manipulating a wide variety of models. The framework uses a hierarchically organized, partitioned, and attributed acyclic graph to represent the semantic as well as mathematical structure of a model. The computer-based environment is evolving via experimental prototypes that provide for ad hoc query, immediate expression evaluation, solving simultaneous systems, and optimization.

If successful, Structured Modeling will enable model-based work to be done with greater productivity and acceptance by non-specialists, will exploit important developments in small computers, and will cross-fertilize management science/operations research, artificial intelligence, database management, programming language design, and software engineering.

This paper is an introduction and status report on a long term project. The presentation is based largely on examples; rigorous development and details are left to a series of technical reports.

TABLE OF CONTENTS

1. INTRODUCTION

Structured modeling is an approach to modeling and also to the design of computer-based modeling systems quite different from current ones. This section motivates the need for structured modeling by considering some of the problems and opportunities presently facing the management science/operations research (MS/OR) community. These suggest some desirable features for future modeling systems. Those features are the main objectives of structured modeling.

1.1 *Problems and Opportunities Facing MS/OR*

The two problems and four opportunities discussed below are among the more important ones confronting the MS/OR community.

Low Productivity

Doing MS/OR tends to be a low productivity activity. Even seasoned practitioners are repeatedly surprised by how much effort is needed to achieve useful results.

A contributing factor is that at least three distinct representations typically are used for each model: a "natural" representation suitable for communication with people (often managers) without special training in MS/OR, a mathematical representation suitable for analytical use, and a computer-executable representation (see, e.g., Fourer <1983>). Such multiple representations are inefficient by virtue of their redundancy, are susceptible to inconsistency, and they demand too many different skills to complete even small projects.

A second factor contributing to low productivity is that interfacing models with advanced solvers (especially optimizers) traditionally has been a laborious task requiring specialized skills. Typically the burden falls on the user to present the model at hand in a format acceptable to the chosen solver. Interface standards are sorely needed. The only one commonly used, the MPS standard for linear programming, is ancient and not very suitable for modern mathematical programming systems.

A third factor is that most modeling software addresses just one among the many kinds of models that arise -- e.g., just linear programs, or just multi-period financial models. Such software is awkward at best and unusable at worst when, as is increasingly necessary, models of different kinds must be integrated in order to address issues of importance. There is a need for modeling software of wider applicability.

A fourth factor contributing to low productivity is that available modeling software typically caters to just one or two of the many phases of the total life-cycle associated with model-based analysis and systems. Some of the more important phases are: determine requirements, design, build, test, use, revise, maintain, document, explain, analyze results, report findings, and evolve. Most MS/OR practitioners are forced to piece together a patchwork quilt of tools to deal with these various phases as they arise over the life of a project. Modeling environments with greater life-cycle scope are needed (Gass <1987>).

Poor Managerial Acceptance

A second and much lamented problem facing MS/OR is that managers and policy makers call for model-based assistance too infrequently.

One reason for this is that MS/OR practitioners and their work often are incomprehensible to non-specialists. To the extent that practitioners are poor communicators or techno-centric instead of problem-centric, managers perceive insufficient empathy and business understanding and hence turn elsewhere for help.

A related reason is that even technically successful MS/OR work can make managers feel <u>less</u> powerful rather than more so. This occurs whenever a manager becomes dependent on the MS/OR practitioner -- as usually happens when the manager does not really understand the model or how it can be used to arrive at conclusions of practical interest. The natural response to this kind of dependency is to avoid it.

These perennial problems are counterbalanced by perennial opportunities often recited by the MS/OR faithful. In addition, there have recently emerged certain new opportunities each of which, if properly exploited, has the potential to exert an influence of historic proportions.

Desktop Computing Revolution

One important opportunity is the desktop computing revolution. This rapidly evolving technology offers numerous possibilities for doing MS/OR more productively, and communicating and delivering MS/OR in ways that managers and policy makers are more likely to accept (Gass, Greenberg, Hoffman and Langley <1986>). The next generation of desktop machines promises to remove many of the remaining barriers to desktop implementation (Crecine <1986>).

Emerging Foundations of Modeling

Another opportunity is that modeling has, in recent years, become an active subject of study in its own right for researchers in several fields: database management (see, e.g., Brodie <1984> and Tsichritzis and Lochovsky <1982> on "data models"), programming language design (see, e.g., Horowitz <1984> and Shaw <1984>), and artificial intelligence (see, e.g., Brachman and Levesque <1985> and Mylopoulos and Levesque <1984> on "knowledge representation"). A particularly noteworthy development is the "conceptual modeling" movement (Brodie et al <1984>), which attempts to synthesize what is known about modeling issues common to all three fields.

It is both surprising and inviting that these fields make virtually no reference to the literature of MS/OR or its closely kindred fields. MS/OR, for its part, traditionally has taken modeling for granted as whatever anyone wants to posit within the conventional languages of mathematics, and thus has failed to develop any coherent modeling theories of its own. The development of new foundations for analytical modeling is long overdue and has many good ideas to draw upon from the three fields mentioned.

Progress in Database Management

A third opportunity for MS/OR is the remarkable flowering of the field of database management during the last decade, especially the explosive emergence of relational technology, the development of excellent database programs for desktop computers (Krasnoff and Dickinson <1986>), and the evolution of sophisticated query interfaces (Jarke and Vassiliou <1985>). Database systems are natural adjuncts to data-hungry MS/OR software. Data management and flexible retrieval capability are just as important for most MS/OR applications as the functions performed by the solvers toward which the models usually are oriented.

Popularity of Spreadsheet Modeling

A fourth opportunity, not unrelated to the first one, arises from the legions of modeling enthusiasts created by the phenomenal rise of spreadsheet software. Many of these people have the potential to graduate to more sophisticated modeling, and so form a great reservoir of potential demand for MS/OR technology and expertise (Bodily <1986>).

1.2 *The Answer: A New Generation of Modeling Systems*

The problems and opportunities just enumerated call for a new generation of modeling systems with the following desirable features:

(a) a rigorous and coherent conceptual framework for

modeling based on a single model representation format suitable for managerial communication, mathematical use, and direct computer execution

(b) independence of model representation and model solution, with model interface standards to facilitate building a library of models and of easily accessed solvers for retrieval, systems of simultaneous equations, optimization, and other important manipulations

(c) sufficient generality to encompass most of the great modeling paradigms that MS/OR and kindred model-based fields have developed for organizing the complexity of reality (activity analysis, decision trees, flow networks, graphs, markov chains, queueing systems, etc.)

(d) usefulness for most phases of the entire life-cycle associated with model-based work

(e) representational independence of general model structure and the detailed data needed to describe specific model instances

(f) desktop implementation with a modern user interface (e.g., visually interactive, directly manipulative, syntactically humane, and with liberal use of graphics and tables)

(g) integrated facilities for data management and ad hoc query in the tradition of database systems

(h) immediate expression evaluation in the tradition of desktop spreadsheet software.

Features (a) through (d) address, respectively, the four contributing factors listed earlier for low productivity. Feature (c) also helps productivity by reducing learning time in situations where multiple models must be maintained. Productivity is further enhanced by feature (e), which facilitates reusing the same general model structure in different specific applications.

Feature (a) should help to overcome poor managerial acceptance to the extent that it succeeds in facilitating managerial communication. Feature (e) should also facilitate communication, for general model structure is free of distracting detail. Feature (f) not only can lead to improved managerial acceptance, but may even be a prerequisite for it.

Features (f), (a), (g), and (h) respectively address the four opportunities listed earlier. In addition, feature (e)

is one of the recurring themes of the second opportunity (the emerging foundations of modeling). It is one of the pillars of database theory because specific database content changes far more frequently than does database structure.

1.3 *Structured Modeling*

Structured modeling aims to provide the foundation for a new generation of modeling systems with all of the features listed in Section 1.2. It also aims to influence how model-based work is carried out using more conventional modeling systems.

The formal framework of structured modeling is based on discrete mathematics. It uses a hierarchically organized, partitioned, and attributed acyclic graph to represent a model or a model class. Particular attention is given to representing semantic as well as mathematical structure, and to compatibility with four of the most fundamental manipulations applied to models: retrieval, expression evaluation, solving a simultaneous system, and optimization.

At the core of structured modeling is the notion of a definitional system, that is, a system of definitions of all of the elements comprising a "model". The definitions have some special properties: they are typed (there are five types), correlated (interdependencies are explicit), and certain of the types are value-bearing. Moreover, the definitions are grouped by definitional similarity, the resulting groups are organized hierarchically by conceptual similarity, and the whole system of definitions must be free of circularity.

This kind of definitional system turns out to be widely applicable within model-oriented fields such as MS/OR/DSS (for finance, logistics, marketing, production, and other application areas), information systems, economics, and engineering. Thus structured modeling ideas have the potential for wide adoption.

This kind of definitional system also turns out to have deep connections to formalisms used in artificial intelligence, database management, programming language design, and software engineering. These connections invite cross-fertilization among these fields from the modeling perspective.

Structured modeling ideas may be useful even if structured modeling software is not available or is not selected for use. Usually there are many opportunities in the context of conventional modeling systems to use some of the modeling concepts, constructs, and guidelines that comprise the structured modeling approach. Some of the guidelines for "good" modeling associated with structured modeling are: (1) incorporate important data development processes directly into the model, (2) document definitional interdependencies, (3) use stepwise refinement,

(4) compose models from validated submodels, and (5) exploit parallel structure.

Finally, it should be noted that structured modeling lays the foundation for a unified theory of model aggregation. This was the original need that led to the development of structured modeling. A draft research monograph on aggregation is at an advanced stage of preparation, but is now suspended pending completion of basic work on structured modeling.

Related Modeling Approaches and Systems

Structured modeling has benefited significantly from ideas introduced by or embodied in other modeling approaches and systems. Numerous opportunities remain for cross-fertilization. We consider briefly some of the principal categories of related approaches and systems from the point of view of the eight desirable features. A companion paper gives further details (Geoffrion <1986b>).

Names of commercially available software packages are given in italics. Their vendors are listed in a separate reference section following the bibliography.

Many attempts have been made to make mathematical programming systems easier to use by orienting them more toward modeling and less toward the optimizers around which they are built. Two standouts are GAMS (Bisschop and Meeraus <1982>, Kendrick and Meeraus <1987>) and PLATOFORM (Palmer <1984>). Others are AMPL (Fourer, Gay and Kernighan <1987>), CAMPS (Lucas and Mitra <1985>), EMP (Schittkowski <1985>), GXMP (Dolk <1986b>), *LINDO*, LPMODEL (Katz, Risman and Rodeh <1980>), MLD (Burger <1982>), and *PAM*. As a group, the greatest strength of these systems -- their ability to raise the productivity of optimization applications -- is perhaps also their greatest weakness in that they are wedded to one particular modeling paradigm (contrary to desired feature (c)). They also lack integrated facilities for ad hoc query and immediate expression evaluation.

Financial planning systems are designed primarily to support the preparation of business analyses and reports based on the spreadsheet modeling paradigm, that is, based on named rows and on columns that usually correspond to successive time periods. This paradigm turns out to be of surprisingly general applicability (Bodily <1986>, Plane <1986>). Unlike the leading desktop spreadsheet packages, these systems use a simple declarative language to specify the spreadsheet, and they automatically attempt to solve any simultaneous equations that may be implicit in the spreadsheet. The dominant package in this class is the mainframe system *IFPS* (also available in desktop versions). A version is even available with integrated optimization capability (Roy, Lasdon and Lordeman <1986>). As a group, financial planning systems offer high productivity, support for multiple life-cycle phases, and good managerial understandability within

their intended domain of application. Their main weaknesses are in the areas of compatibility with modeling paradigms other than the spreadsheet, independence of general model structure and detailed data, and integrated facilities for ad hoc query.

Database management systems usually are not thought of as "modeling" systems by the MS/OR community, but this is a mistake. All are based on one or another data model (Tsichritzis and Lochovsky <1982>). A recent survey (Krasnoff and Dickinson <1986>) lists 56 desktop relational database systems. As a group these systems are, of course, strong on database functions. But they are weak on compatibility with non-database modeling paradigms and on provisions for accessing non-database solvers for simultaneous equations and optimization.

Integrated multi-function desktop productivity software, of which *Framework* is a current example, provides word processing, spreadsheet modeling, some database capabilities, business graphics, a built-in programming language, and other useful functions. One recent package, *Guru*, even includes an expert system shell. These systems can be excellent productivity tools for a wide variety of tasks, including many kinds of modeling. However, it may not be appropriate to think of them as modeling systems in a true sense because they do not offer a coherent conceptual framework for modeling. How they fare by features (b) through (e) depends largely on how they are used. They do possess features (f), (h), and perhaps (g).

A related approach is a modeling environment based on loosely integrated utilities for data extraction, large file manipulation, data management, solving simultaneous systems, and other functions arising in model-based work. A nice example is ANALYTICOL at AT&T Bell Laboratories (Childs and Meacham <1985>). It fares well by features (c), (d) and perhaps (g), and not so well by features (a), (h), and perhaps (f). How it fares by features (b) and (e) depends almost entirely on how it is used.

The above categories do not exhaust the landscape of modeling approaches and systems. For example, there are discrete event simulation frameworks and languages (e.g., Markowitz <1979>, Oren, Zeigler and Elzas <1984>); various knowledge representation approaches from artificial intelligence (Brachman and Levesque <1985>), among which the closest to structured modeling appears to be the "conceptual graph" formalism of Sowa <1984> within the general category of semantic networks; and the novel approach of Jones <1985> to graphical modeling systems based on attributed graph grammars, which can capture general model structure at a level similar to that used in structured modeling. Each of these is discussed at some length in Geoffrion <1986b>. In addition, of course, there are numerous application-specific packages which, by definition, have conceptual frameworks of narrow applicability and thus lack feature (c).

This brief review of related modeling approaches and systems is not intended to show that structured modeling's predecessors are inferior because none exhibits all eight desirable features. Rather, the review is intended to show that structured modeling aims at an apparently vacant niche, and to suggest that there are benefits to be gained (in both directions) from studying structured modeling in the context of alternative approaches and systems.

1.4 *Organization*

Section 2 contains an introduction to the basic ideas of structured modeling, an example, an explanation of how these ideas fit into the world of model-based work, and a discussion of the prospects for achieving a new generation of modeling systems of the type envisioned in Section 1.2. Rigorous definitions and selected technical results on the structured modeling framework are deferred to the Appendix.

Section 3 exemplifies three of the ways in which structured modeling can be used: for top-down model design by stepwise refinement, for integrated modeling in the sense of unifying two or more distinct models in a coordinated way, and for clear communication and documentation. All five of the modeling guidelines listed in the previous subsection are illustrated along the way.

Section 4 describes structured modeling systems in general and three development prototypes in particular, with emphasis on the *Framework*-based implementation presently in progress at UCLA.

Section 5 discusses opportunities for further research and development. Many are inspired by important parallels between issues in structured modeling and similar issues in database management, programming language design and software engineering, and artificial intelligence. The opportunities are collected into four main categories: the structured modeling framework itself, designing a model within the framework, computer implementation design, and model management systems.

Finally, Section 6 offers some closing comments.

This paper makes few assumptions concerning the reader's background beyond general familiarity with MS/OR and tolerance for basic terminology drawn from graph theory and a few other parts of discrete mathematics. Familiarity with elementary relational database ideas will also be helpful (e.g., Date <1981>).

Because of the introductory nature of this paper, it is necessary to defer numerous details and related developments to a forthcoming series of technical reports.

2. BASICS OF STRUCTURED MODELING

The structured modeling framework has three levels: elemental structure, generic structure, and modular structure. Each is defined in turn, followed by an illustration (as well as other ideas) based on a classic MS/OR application, the feedmix model. Then the role of these modeling ideas is discussed in a broader context, followed by an assessment of their potential as the basis for a new generation of modeling systems of the type advocated in Section 1.2.

2.1 *Basic Definitions*

The definitions given here are, for the most part, informal. Rigorous versions are given in the Appendix of this paper, but a forthcoming technical report must be consulted for a complete development. A detailed example is given in the next subsection for all of these definitions.

The reader is invited at this point to test the author's claim that structured models are easy to understand. The severest possible test would be to examine a structured model prior to any study of structured modeling concepts. Doing this will not only help the reader judge the claim, but also aid digestion of the definitions. So please spend a few minutes examining, in this order, Figure 5 (the general structure of the feedmix model), Figure 6 (sample data for a particular instance), Figure 3 (a diagram of definitional dependencies at a more aggregate, dimensionally independent level), and Figure 4 (a way of organizing the essential concepts of the model).

Elemental Structure

Structured modeling views a model as being composed of discrete *elements*. The central notion is that each element has a definition in which the element's existence is either postulated as a primitive of the model, or postulated in terms of other elements whose definitions have already been given.

Elemental structure aims to capture all of the definitional detail of a specific model instance. It can be viewed in terms of a directed graph of elements (nodes) and "calls" (arcs). Each *call* represents a definitional reference, that is, the participation of one element's definition in the definition of another. The head node of each arc is the calling element and the tail node is the called element. There are five *types* of elements, some of which have a *value*:

1) *primitive entity* elements have no associated value and generally represent things or concepts postulated as primitives of the model (e.g., protein as a nutrient);

2) *compound entity* elements have no associated value and generally represent things or concepts that are defined in terms of other things or concepts (e.g., a "link" in a transportation system defined in terms of a certain plant and a certain customer);

3) *attribute* elements have a constant value and generally represent properties of things or concepts (e.g., a minimum daily requirement in grams associated with protein);

4) *function* elements have a value that is dependent according to a definite rule on the values of called elements, and generally represent calculable properties and more complex aspects of models (e.g., the total annual cost associated with inventories);

5) *test* elements are like function elements except that their value must be either True or False (e.g., whether the minimum daily requirement level for protein is met).

The graph is assumed to be acyclic because it is possible and desirable to avoid circular definitions. If the graph is to represent the entire elemental structure, then it must also be attributed. Attributes must be associated with its nodes and arcs to represent (i) the values of non-entity elements, (ii) the rules by which the values of function and test elements are calculated, and (iii) an order for the inbound arcs at each node.

Generic Structure

Generic structure aims to capture the natural familial groupings of elements. Mathematically, this is accomplished by partitioning all elements of a given type into *genera*, each of which is a cell of the partition. Thus each genus comprises elements of the same type (e.g., the collection of all primitive entity elements representing nutrients).

Not every possible partition by type is allowed. It must satisfy a property called *generic similarity*, which means roughly that every element in a genus calls elements in the same foreign genera (e.g., every element in the minimum daily requirement genus makes reference to some element in the nutrient genus). This property seems to hold for all sensible partitions and is essential in order to prove certain desirable properties of the structured modeling framework.

Modular Structure

Modular structure aims to organize generic structure hierarchically to the extent that this seems appropriate and useful. The basic idea is to group genera into conceptual units called *modules* according to commonality or semantic relatedness, then to group these modules into higher order modules, and so on (e.g., the nutrient genus and the minimum daily requirement genus might be grouped together into a "nutrient data" module). This enables the complexity of a model to be managed in terms of higher order abstractions.

Mathematically, modular structure is a rooted tree whose root represents the entire model and whose terminal nodes correspond 1:1 with the genera. All other nodes are modules representing conceptual units comprising their descendent genera.

Not every possible modular structure is allowed. It must admit an indented list representation with no forward references, that is, the genera must be listed in such an order that no element in a genus ever calls an element in a genus that is farther down the list. A modular structure that satisfies this qualification is called *monotone*, and its indented list representation is called a *modular outline*.

In practice it is easy and natural to define a monotone modular structure. But if for some reason it is desired to ignore modular structure or to postpone its design, then it is always possible to posit the trivial monotone modular structure in which there are no modules other than the root and all genera are ordered according to a topological sort of the genus graph (defined in Section 2.2). Such a sort is easy to perform. There is also an efficient procedure for finding a monotone order, if one exists, for any given modular structure.

Structured Model

Finally we can define a *structured model* as (a) an elemental structure together with (b) a generic structure satisfying similarity and (c) a monotone modular structure.

It should be noted that the acyclicity assumption on elemental structure and the closely related monotonicity assumption on modular structure do not necessarily preclude representing models with simultaneity or recursion. In all realistic cases examined to date, simultaneity and recursion can be dealt with in a natural way without violating these assumptions. Sometimes this involves switching to an equivalent representation of some model feature, sometimes it involves modeling in such a way that a "solver" external to the model carries the burden (as by solving a system of simultaneous equations), and sometimes it simply involves recognizing that simultaneity or recursion exists in a

way that does not impinge on the acyclicity or monotonicity assumptions. These assumptions play a key role in the theory and application of structured modeling.

It should also be noted that structured modeling is not limited to static models. Dynamic models with discretized time can be accommodated by introducing a primitive entity genus with as many elements as there are time instants or time slices, and dynamic models with continuous time often can be accommodated by allowing the values of attribute and function elements to be functions of time.

Model Schema

Up to this point we have been concerned with specific model instances. However, the focus of applied modeling work is very rarely on a single model instance. Nearly always it is on an entire class of similar instances. Therefore it is appropriate to formalize the notion of a class of "similar" structured models. That is the purpose of the concept of a *model schema*.

Informally speaking, a model schema is any class of structured models whose modular outlines all can be placed in 1:1 correspondence in a way that is consistent with modular structure, with generic structure, and with the intended meaning of the models.

2.2 *Example and Additional Concepts*

The feedmix problem can be found in the linear programming chapter of most basic textbooks on MS/OR. We use it to illustrate the concepts just defined and to introduce some important additional ideas and notational conventions.

Element Graph

Figure 1 is the *element graph* for a simple model with two nutrients and two materials from which feeds are blended. It is the directed graph of elemental structure without any annotations to indicate node or arc attributes. Figure 1 does, however, employ informal annotations to indicate node interpretation and type. Node type is indicated here by the shape of the symbol used: squares for primitive entity elements, circles for attribute elements, triangles for function elements, and hexagons for test elements. There are no compound entity elements. Recall that an arc represents a call of the tail element by the head element.

The process of calculating the values of all function and test elements in their natural topological order is called *evaluation*. Such an order always exists by virtue of the acyclicity assumption. For Figure 1, evaluation can be accomplished by proceeding from bottom to top.

Genus Graph

There is an obvious grouping of elements. Primitive entity elements are partitioned into nutrient elements (the NUTR genus) and materials elements (the MATERIAL genus). Attribute elements are partitioned into minimum daily requirement elements (the MIN genus), elements specifying how much of each nutrient is in each pound of material (the ANALYSIS genus), quantity elements (the Q genus), and unit cost elements (the UCOST genus). Function elements are partitioned into the elements that calculate the achieved nutrition levels (the NLEVEL genus) and the total cost element (the singleton TOTCOST genus). Finally, the test elements are all left together in a single genus (T:NLEVEL).

These partitions are shown in Figure 2. The informal definition of generic similarity clearly holds by inspection of the incoming arcs to the elements in each genus.

The graph theoretic condensation of an element graph according to such partitions is called the *genus graph;* see Figure 3. It is more convenient than the element graph for most purposes because it is dimension independent. For example, Figure 3 does not depend on how many nutrients or materials there may be. It can be shown that the genus graph is always acyclic when the element graph is finite and acyclic and generic similarity holds.

Modular Tree

Several plausible hierarchical organizations of generic structure are possible, including this one: group the genera NUTR and MIN together in a module whose interpretation is "nutrient data", and group MATERIAL, UCOST, and ANALYSIS together in another module whose interpretation is "material data". This modular structure is represented by the *modular tree* shown in Figure 4. Note that there is no mathematical connection between the arcs of the genus graph and those of the modular tree.

Figure 4 also includes an indented list representation of the tree. It can be seen from Figure 3 that there are no forward references in this list, and so it is a bona fide modular outline.

Non-Graphical Notation

Element graphs, genus graphs, and modular trees offer a practical vehicle for expressing elemental, generic, and modular structure, to say nothing of expressing a model schema, only if supported by a software system with advanced graphics capabilities. A less demanding alternative is to express generic and modular structure and model schemata by a text-based "schema",

and to express elemental structure by such a schema together with a collection of "elemental detail tables". Both concepts will be explained in some detail.

The particular notational conventions used here for text-based schemata and elemental detail tables constitute only one possible solution to the problem of designing a practical notation for structured models and model schemata. They have certain advantages, as will become evident, but do not "define" structured modeling in any sense. Others may wish and, indeed, are encouraged to propose other notational conventions to embody the core concepts of structured modeling described informally above and formally in the Appendix.

Schema

Figure 5 gives a text-based *schema* that expresses the generic and modular structure of the feedmix example. It also represents a model schema in the sense defined earlier, that is, an entire class of structured models for feedmixing whose modular outlines are all 1:1 with one another in a consistent way -- in fact, they are all identical.

Figure 5 uses a format and syntax that is detailed in a forthcoming technical report. An acquaintance with this schema will be a big step toward understanding schemata in general. Thus we give an overview of schema format and syntax followed by a narrative interpretation of the particular schema at hand.

1. A schema is composed of *paragraphs*, one for each line of the modular outline and indented in exactly the same way. There are two kinds of paragraphs: *module paragraphs* describing modules, which always begin with the module name, and *genus paragraphs* describing genera, which always begin with the genus name.

2. *Module names* and *genus names* are unique and capitalized. The former always begin with an ampersand (&) for quick recognition, while the latter always begin with a letter of the alphabet.

3. Every paragraph consists of two parts: a *formal* part followed by an *interpretation* part. The interpretation part is distinguished in Figure 5 by being printed in italics (a special separation character is used when the schema is prepared using a single-font editor). The syntax of the interpretation part is essentially unrestricted, although the style followed here is recommended, including: introduce an underlined, capitalized, unique *key phrase* in each paragraph and capitalize this phrase at each subsequent use. The purpose is to provide easily readable documentation. The formal part of a module paragraph consists only of the module name. The syntax of the formal part of a genus paragraph

is the subject of the remaining comments.

4. The formal part of a genus paragraph always includes a *type indicator* (/pe/, /ce/, /a/, /f/, or /t/) to indicate element type. The indicator /va/, for *variable attribute*, can be used in place of /a/ for an attribute genus when its values are discretionary and hence likely to change or to be placed under solver control.

5. The genus name in a non-/pe/ genus paragraph is always followed by a *generic calling sequence* in parentheses that identifies all of the elements which participate in the definition of a typical element. The syntax of generic calling sequences is designed so that the generic similarity property holds.

6. The type indicator in a genus paragraph usually is followed by an *index set statement* that specifies the element population of the genus. If omitted, then it is understood that every possible element exists.

7. The index set statement of an attribute genus paragraph usually is followed by a *range statement*, announced by a colon, that specifies the allowable values for the elements of the genus.

8. The index set statement of a function or test genus paragraph is always followed by a *generic rule*, announced by a semicolon, that specifies how the element values are to be calculated.

9. Every genus that can have more than one element is fully indexed. An index is never "dummy", but is always a specific lower case letter uniquely associated with the genus that introduces it. A genus that introduces an index is said to be *self-indexed*, and the index is given immediately after the genus name in its genus paragraph. A genus that does not introduce an index is *externally indexed* unless it must be a singleton. It is indexed by the free indices in its calling sequence.

The full syntax associated with items 5-8 is a lot richer than is apparent from the simple example presented in Figure 5. Nevertheless, that schema suffices for illustrative purposes.

The first paragraph simply says that there is a module named &NUT_DATA having to do with nutrient data. Indentation reveals that there are two genera in this module, NUTR and MIN.

The second paragraph says that there is a primitive entity genus named NUTR whose elements are indexed by i. It does not say how many elements are in the genus or what any of them are. That is the job of the elemental detail tables. NUTR is self-indexed.

The third paragraph says that there is an attribute genus named MIN whose typical element MINi calls element NUTRi. The index set statement (NUTR) says that MIN has one element for every element of NUTR. The range statement says that all elements have nonnegative real values. MIN is externally indexed (by i), as are all subsequent genera except MATERIAL (which is self-indexed by m) and TOTCOST (which has no indices at all).

The next three paragraphs are similar to the first three.

The paragraph for the attribute genus ANALYSIS has a typical element ANALYSISim that calls elements NUTRi and MATERIALm. The index set statement says that ANALYSIS has an element for every nutrient-material combination, and the range statement says that all elements have nonnegative real values.

Consider the first function genus. Its typical element NLEVELi calls ANALYSISim for all m (this is indicated by the dot in place of the second index of ANALYSIS in the calling sequence), and also Qm for all m (note that Q appears in the calling sequence with none of its indices -- alternatively, "Q." could have been used). The generic rule says that NLEVELi is calculated by summing the product of ANALYSISim and Qm over all m.

The other function genus, TOTCOST, must be a singleton because there are no free indices in its calling sequence. It calls all elements of genera UCOST and Q. The meaning of the generic rule should be evident.

The test genus has typical element T:NLEVELi, which calls the corresponding element of NLEVEL and the corresponding element of MIN. The index set statement stipulates that there is a T:NLEVEL element for every element of NUTR. The generic rule stipulates that T:NELEVELi has value True if and only if the value of NLEVELi is greater than or equal to the value of MINi.

Elemental Detail Tables

The purpose of *elemental detail tables* is to describe a particular instance of the general class of models represented by a schema. The skeletal structure of elemental detail tables is automatically determined from the schema according to rules given in a forthcoming technical report.

Figure 6 gives the six elemental detail tables for the schema of Figure 5.

The first table, named NUTR, lists the *identifiers* of the nutrients and the minimum daily requirement value for each. It also provides for an interpretation of the nutrient identifiers.

In general, the identifiers introduced by a self-indexed genus can be any unique names, and the interpretation column helps document the modeler's intentions.

The second table performs a function similar to the first, but for materials instead of nutrients.

The ANALYSIS table gives the analysis values for all elements in the ANALYSIS genus. The other tables are equally obvious.

2.3 *Models, Problems, and Solvers*

The foregoing has focused entirely on modeling. What about the things one does with models?

We make a sharp distinction between a "model" as an abstraction of reality, the "problems" or "tasks" one poses in terms of a model, and the "solver" used to solve a problem or carry out a task. For example, consider the feedmix model. The usual problem associated with this model is to find values for the QUANTITY elements so as to minimize the value of the TOTAL COST element subject to the values of all NUTRITION TEST elements being True. The type of solver most appropriate to this kind of problem is well known, namely one that implements an algorithm for linear programming.

Optimization is not the only important kind of problem that one might wish to pose in connection with a model. Two others are ad hoc queries aimed at retrieving information about the model, and finding values for selected attribute elements so that certain test elements are True. The former requires a type of solver sometimes called a query processor in the field of database management, and the latter often requires an equation solver. Many other important kinds of problems and tasks could be identified, such as drawing logical inferences (which requires a type of solver sometimes known as an inference engine in the field of artificial intelligence).

Structured modeling provides a framework for modeling within which various problems and tasks can be posed precisely and naturally. It does not provide a framework that directly supports the algorithmic aspects of solvers. Solver design and implementation is an entirely distinct area to which modeling bears a client relationship.

A structured modeling system should, however, make provision for invoking solvers of various kinds. These can be thought of as residing in a "solver library", where they are conveniently available for use whenever needed.

2.4 *Prospects for Achieving a New Generation of Modeling Systems*

Now that the basics of structured modeling have been explained, it is appropriate to consider the potential of these ideas as a basis for the new generation of modeling systems advocated in Section 1.2. The discussion is organized according to the eight desirable features given there.

Feature (a): rigorous modeling framework with a single model representation suitable for managerial communication, mathematical use, and direct computer execution. The structured modeling framework described previously is rigorous and offers a choice of two notational styles: one based on attributed graphs (the element graph, genus graph, and modular tree) and one comprising a text-based schema together, if a specific model instance is required, with elemental detail tables. A structured modeling system could be based on either one or a combination of these styles.

Suitability for managerial communication is discussed in some detail in Section 3.3 where, among other points, it is noted that genus graphs are particularly attractive devices for managerial communication.

Suitability for mathematical use depends on what kind of mathematics one wishes to apply. Obviously there is no issue with respect to graph theoretic mathematics. But there may be for other kinds of mathematics. Probably the most important kind, at least in MS/OR, is ordinary algebra with indexing over sets of similar mathematical objects. One of the reasons for introducing the text-based schema notation was to enable this kind of mathematics (see especially items 5, 6, 8, and 9 in Section 2.2).

Direct computer executability is possible if due care is exercised in designing the notational conventions supported by a structured modeling system. Standard compiler technology (e.g., Aho, Sethi, and Ullman <1986>) can be used to digest schemata like Figures 5, 7, and 10 because they can be (and, in fact, are) written in a context-free language. Graph-based notations probably can be digested with the help of graph grammars (see Jones <1985>).

Feature (b): model/solver independence with interface standards and provision for libraries of models and solvers. It is clear from Section 2.3 that structured models are entirely separate from any solver that may be invoked on them. Structured models can be kept in libraries and so can solvers. The design of interface standards that facilitate coupling solvers to models is a technical challenge that seems not overly difficult because all models can be represented using the very same formalism. The knotty problem of interface standards can be dealt with by inverting the usual approach: instead of living with

multiple solver-oriented standards for modelers to write to, provide a single model-oriented standard for solver and modeling system technicians to write to.

Building a library of models, especially of model classes, facilitates reusing old models in new situations. A similar point holds with respect to building a library of solvers. An organization that undertakes modeling efforts frequently could achieve a significant productivity gain from such reuse. This suggests that it could be worthwhile for a group of similar organizations to collaborate on a joint library of models and solvers, or for a library of generic applicability to be built for a specific functional area. The latter approach is being implemented for marketing at Purdue, where a major commercial database as well as traditional descriptive and normative marketing models are being cast in structured form (Wright <1986>).

Feature (c): generality. The generality of structured modeling follows from the fact, explained in Section 1.3, that it formalizes the notion of a definitional system as a way to describe models -- both model classes and particular model instances. Structured modeling does not aim to offer a modeling paradigm in the usual sense, but rather a lingua franca within which models from a wide variety of paradigms can be expressed.

The three simple models given in this paper, all drawn from MS/OR, do not begin to illustrate the generality of structured modeling. An extensive and more suggestive collection of structured models is in preparation; it covers a wide variety of applications to business, database management, economics, engineering, MS/OR, and various other application areas.

Feature (d): life-cycle orientation. The life-cycle of a modeling application goes from the initial feasibility analysis to the final completion of the original objectives. Examination of the many stages in between shows that true life-cycle orientation requires what might best be thought of as an interactive work "environment". This environment should support not only models and solvers as discussed previously, but should also offer a variety of utilities needed for communication, organizing things and ideas, and for different kinds of ancillary quantitative analysis. This poses a considerable challenge for the design of structured modeling system implementations.

Feature (e): general structure/detailed data independence. The notion of general structure is captured in structured modeling by the modular and generic structure formalisms. The notion of detailed data is captured by the notion of elemental structure. The former can be expressed by a text-based schema, and the latter by elemental detail tables.

Feature (f): desktop computer based with a modern user interface. There are two primary challenges: i) performance difficulties when the number of elements gets large or the user

interface gets sophisticated, and ii) access to mainframe data and programs. The next generation of personal workstations (e.g., Crecine <1986>) should provide sufficient resources to deal with the first challenge for models of at least moderate size, and mainframe links are progressing rapidly (e.g., Derfler <1986>).

Feature (g): integrated ad hoc query capabilities. It can be shown under mild assumptions that elemental detail tables (proposed for non-graphical notation) can be viewed as a relational database in third normal form or higher (see, e.g., Ullman <1982>). The primary key columns are the ones to the left of the vertical double lines; these we call the *stub columns*.

This is an important result because it establishes a bridge between structured modeling and the relatively mature field of relational databases. One useful consequence is that a strong point of departure is immediately available for the development of query languages for ad hoc retrieval and for implementations of structured modeling systems.

Feature (h): resident expression evaluation capability. This requirement is attainable using spreadsheet technology and extensions thereof. It has the potential for considerable efficiency because of element graph acyclicity (the required topological sort can be done once and for all for a given schema).

It is reasonable to conclude that all of the features defining a "new generation" are achievable by a properly designed and implemented structured modeling system. They all have, in fact, been achieved to some degree by prototype implementations (see Section 4).

Of course, the simultaneous achievement of all these features is likely to exact a price in terms of complexity and performance relative to systems with more modest ambitions, or that have a narrower domain of application. Will structured modeling systems be simple enough to be usable by application domain experts? Will they be efficient enough for production applications as well as prototyping? Will they be able to compete with more highly specialized systems? The answers to such questions must await further progress in computer implementation.

3. SOME USES OF STRUCTURED MODELING

This section illustrates some of the ways in which a structured modeling system could be used: to do "top-down" model design by stepwise refinement, to do "integrated" modeling, and for communication and documentation.

3.1 *Top-Down Model Design*

Top-down design is a time-honored concept that has been used with success by engineers, computer software designers, and probably by almost every profession concerned with undertakings of great complexity. For present purposes, "top-down design" means *stepwise refinement based on a hierarchical view of complexity*. The rationale for this approach is to attempt to get the "big picture" right at the outset with a minimum of distracting or inessential clutter, and then to add detail in stages that take advantage of previously established perspective. The overall effect is one of hierarchical decomposition of the complexity dimension.

Structured modeling provides a hospitable framework within which to develop top-down ideas because modular structure *is* hierarchy, and because generic structure usually lends itself conveniently to refinement.

Example

We illustrate top-down model design using another classical model of management science as the point of departure, namely the economic order quantity model with multiple (independent) items. Figure 7 presents a schema for this model along with elemental detail tables containing sample data. The reader should have no difficulty deciphering this schema based on the explanations given in Section 2.2.

DEMAND RATE, HOLDING COST RATE, and FIXED SETUP COST must, in virtually every real application, be calculated from other data. Thus the schema of Figure 7 is but a "first pass" toward a truly applicable model. It requires further refinement. For the second pass, assume:

(a) DEMAND RATE must be calculated as the sum of demands deriving from several final products

(b) HOLDING COST RATE is the sum of the opportunity cost of capital tied up and the out-of-pocket storage cost

(c) FIXED SETUP COST is the sum of separate costs for the materials and labor consumed.

Consequently, D, H, and F each must be elaborated into an entire module. The result might be as follows for the &ITEMDATA module (the rest of the schema stays the same).

&ITEMDATA *Certain ITEM DATA are provided.*

&D *DEMAND RATE DATA*

FINALPRODp /pe/ *There is a list of FINAL PRODUCTS.*

DPARTIAL(ITEMi,FINALPRODp) /a/ (ITEM)x(FINALPROD) : R+ *Each FINAL PRODUCT contributes a PARTIAL DEMAND RATE (units per year) for each ITEM.*

D(DPARTIALi.) /f/ (ITEM) ; SUMp (DPARTIALip) *Every ITEM has a DEMAND RATE (units per year) equal to the sum of its PARTIAL DEMAND RATES.*

&H *HOLDING COST RATE DATA*

VAL(ITEMi) /a/ (ITEM) : R+ *Every ITEM has a UNIT VALUE (dollars per unit).*

STORAGE(ITEMi) /a/ (ITEM) : R+ *Every ITEM has a STORAGE COST RATE (dollars per unit per year) associated with physical possession.*

H(VALi,STORAGEi) /f/ (ITEM) ; 0.12 * VALi + STORAGEi *Every ITEM has a HOLDING COST RATE (dollars per unit per year) equal to a 12% opportunity cost of capital tied up (calculated on the basis of UNIT VALUE) plus the STORAGE COST RATE.*

&F *SETUP COST DATA*

FMATERIAL(ITEMi) /a/ (ITEM) : R+ *The setup of an ITEM incurs a specific SETUP MATERIAL COST (dollars per setup).*

FLABOR(ITEMi) /a/ (ITEM) : R+ *The setup of an ITEM incurs a specific SETUP LABOR COST (dollars per setup).*

F(FMATERIALi,FLABORi) /f/ (ITEM) ; FMATERIALi + FLABORi *Every ITEM has a FIXED SETUP COST (dollars per setup) equal to SETUP MATERIAL COST plus SETUP LABOR COST.*

Still greater detail can be added at a third pass. Assume:

(d) PARTIAL DEMAND RATES must be built up from demand estimates for FINAL PRODUCTS and the parts explosion

(e) UNIT VALUE must be assembled from its major components

(f) SETUP LABOR COST must be constructed as labor hours times labor rate.

Then **DPARTIAL** becomes this module:

&DPARTIAL *PARTIAL DEMAND RATE DEVELOPMENT*

DFINAL(FINALPRODp) /a/ {FINALPROD} : R+ *Each FINAL PRODUCT has an estimated FINAL PRODUCT DEMAND RATE (units per year).*

BILL(ITEMi,FINALPRODp) /a/ {ITEM}x{FINALPROD} : Int+ *There is a table giving the number of each ITEM in each FINAL PRODUCT; this is called the BILL OF MATERIALS.*

DPARTIAL(BILLip,DFINALp) /f/ {BILL} ; DFINALp * BILLip *Each FINAL PRODUCT contributes a PARTIAL DEMAND RATE (units per year) for each ITEM equal to the estimated FINAL PRODUCT DEMAND RATE times the appropriate BILL OF MATERIALS multiplier.*

In addition, **VAL** becomes a module:

&VAL *UNIT VALUE DEVELOPMENT*

DIRMAT(ITEMi) /a/ {ITEM} : R+ *The value of each ITEM includes a certain amount of DIRECT MATERIAL COST (dollars per unit).*

DIRLAB(ITEMi) /a/ {ITEM} : R+ *The value of each ITEM includes a certain amount of DIRECT LABOR COST (dollars per unit).*

VAL(DIRMATi,DIRLABi) /f/ {ITEM} ; DIRMATi + DIRLABi *Every ITEM has a UNIT VALUE (dollars per unit) equal to the sum of DIRECT MATERIAL COST and DIRECT LABOR COST.*

Finally, **FLABOR** becomes a module:

&FLABOR *SETUP LABOR COST DEVELOPMENT*

SETLABOR /pe/ *There is a labor class known as SETUP LABOR.*

SETRATE(SETLABOR) /a/ {SETLABOR} : R+ *SETUP LABOR is charged at a certain SETUP LABOR RATE (dollars per hour).*

SETHOURS(SETLABOR,ITEMi) /a/ (ITEM) : R+ *The setup of an ITEM requires a specific number of SETUP LABOR HOURS (hours per setup).*

FLABOR(SETRATE,SETHOURSi) /f/ (ITEM) ; SETRATE * SETHOURSi *The setup of an ITEM incurs a specific SETUP LABOR COST (dollars per setup) equal to SETUP LABOR RATE times SETUP LABOR HOURS.*

It is instructive to examine the effect of these stepwise refinements on the genus graph and on the modular outline.

The genus graphs corresponding to each of the three passes are given in Figures 8A, 8B, and 8C. The graphs are drawn so that the additional detail supplied at each pass literally occurs from the top down.

There is a side point to be made here about a common phenomenon that might be called the "data iceberg". Most textbook models represent only the tip of a figurative iceberg, ignoring the great mass of data and associated calculation required underneath to support the tip that it may bask in the light of mathematical analysis and computational solution. Structured modeling can represent this additional detail in an organized way through stepwise schema refinement.

The modular outlines after each of the three design passes are shown in Figure 9. This diagram vividly summarizes the stepwise refinement aspect of the top-down design process. Of course, stepwise refinement may not always lead to such an elegant progression of transformations of the modular outline.

Considering that so many new attribute genera have been introduced, one may wonder whether the final version of the schema is better viewed as a "model" or as a "database"; only a few genera have to do with what is actually required to formulate the classical EOQ cost minimization problem. Structured modeling is indifferent to whether it is representing something that is more like a traditional analytical model or more like a database. The distinction is artificial and will, we hope, gradually dissolve with the advent of more comprehensive modeling approaches like the one proposed.

3.2 *Integrated Modeling*

There are several different things one might mean by "integrated modeling" (Geoffrion <1986c>). Here it is used in the sense of coordinated unification of two or more distinct models. Integration can be across business functions, as when a production model is combined with a distribution model; across

geography, as when regional energy models are combined into a national model; across time, as when a planning model is combined with a scheduling model; or across other dimensions.

Integrated modeling enables results and insights that cannot be achieved by separate models. This becomes increasingly significant for any organization in which modeling has been in use for some time, as the classical approach of many independent applications can be expected to reach a point of diminishing returns. Integrated modeling may even be essential in strategic studies, which usually involve whole systems rather than subsystems, and in very large modeling efforts of any kind, where independent construction, test, and final assembly of component models may be the only practical way to cope with the complexity required of the final model.

For these and other reasons, it is widely believed that integrated modeling is of growing importance (e.g., Harrison and de Kluyver <1984> and Walker <1982>).

Structured modeling provides a natural framework for integrated modeling because it makes explicit the essence of what must be coordinated, namely definitional and computational dependencies among submodels. Moreover, on the basis of limited experience, it appears that the conceptual integrity of the submodels usually can be preserved when integration is done within the structured modeling framework.

Example

A very simple example of integrated modeling is obtained by juxtaposing the well known Hitchcock-Koopmans transportation model with the multi-item EOQ model introduced in the previous subsection. Figure 10 gives a schema for the transportation model, and Figure 11 gives the corresponding elemental detail tables filled in with sample data (not used in this example).

If the transportation model is posed on an annualized basis, solving the usual linear programming problem yields the "optimal" annual flows, but does not prescribe how often shipments should be made or, equivalently, what the shipment size should be. Very frequent shipments are small and thus good for the customer in that they lead to small inventories, but bad in that they are expensive to receive (they require many transactions). Infrequent shipments lead to just the opposite result. The best compromise can be found by solving an EOQ problem for each shipment link.

The situation can be described by two separate structured models. One is the transportation model whose schema is given in Figure 10. The other is the multi-item EOQ model whose schema is as given in Figure 7 with these modifications: (a) replace the **ITEM** genus by a copy of the **LINK** genus (since each transportation link plays the role of an "item"); replace the D

genus by a copy of the **FLOW** genus (since each transportation flow plays the role of a "demand rate"); and rename the **SETUP$** genus to **REC$** to commemorate the reinterpretation of a setup cost as a shipment receiving cost. See Figure 12 for the two genus graphs.

To evaluate costs, one may do the following:

1. Choose values for the **FLOW** variables in the transportation model and evaluate to obtain **$**.

2. Set the **FLOW** attributes in the EOQ model to the same values as the **FLOW** variables in the other model, choose values for the **Q** variables, and evaluate to obtain **TOT$**.

3. Add **$** and **TOT$** to obtain the grand total cost.

If desired, **FLOW** can be chosen in Step 1 by solving the usual linear programming problem, and **Q** can be chosen in Step 2 by solving the usual EOQ problem for its closed form solution.

Sequential rather than simultaneous use of the two models leads, of course, to suboptimization. The two models must be integrated if jointly optimal choices of **FLOW** and **Q** are desired. This can be accomplished by joining the two schemata in this way:

1. Concatenate the two schemata, with the transportation schema coming first (an arbitrary choice).

2. Drop the **LINK** and **FLOW** genera from the second schema, and reroute all calls to them to the **LINK** and **FLOW** genera of the first schema instead.

3. Create a new singleton function genus **TOTCOST** whose purpose is to add the values of **$** and **TOT$**; place it at the very end.

The resulting genus graph is shown in Figure 13, and the modular outline in Figure 14. The global optimization problem can now be posed as choosing values for **FLOW** and **Q** so as to minimize **TOTCOST**.

Notice that the conceptual integrity of each component model is largely preserved.

3.3 *Communication and Documentation*

One of the critical success factors for MS/OR applications is good communication and documentation. Structured modeling offers some attractive possibilities in this area.

Perhaps the most important feature of structured modeling in this regard is the completeness and readability of model schemata and elemental detail tables (see Figures 5-7, 10, and 11). If well executed, they are suitable with only minor adaptation for nontechnical as well as technical audiences. A simplified version of a schema, called the *natural language summary*, is suitable for even the least technical of audiences. It abbreviates the formal part of each genus paragraph to just the genus name with free indices suffixed. See Figure 15 for an example. Schemata and natural language summaries have the appealing property of providing a dictionary of model parts that is without forward references.

Thoughtfully designed modular structure contributes greatly to the readability of a schema, although none of the simple models used in this exposition is substantial enough to exhibit the richness of this concept as a means of organizing complexity. A more realistic example is given in Figure 16, which is based on a capital expenditure planning application done for a telephone company (Geoffrion <1986a>). Only part of the modular tree is shown; it goes to a depth of seven levels. Figure 17 gives a partial natural language summary.

Figures 16 and 17 also illustrate a useful technique for constructing "views" of a model tailored to the needs of specific audiences: prune away subtrees, but do not separate siblings. Figure 16 exhibits two such views, one used to design a series of executive seminars at the vice presidential level of telephone operating companies, and the other used to design a briefing of budget directors. Both views hide unwanted detail. The first view is also used to organize the main managerial documentation for the capital expenditure planning system; in fact, Figure 17 actually is an excerpt from this documentation.

The genus graph (see Figures 3, 8, 12, and 13) holds promise as a communication and documentation aid. This is confirmed by the fact that many authors have independently invented closely related graphics for expository purposes. An example adapted from an introductory quantitative methods text is shown in Figure 18. Many other examples can be found in articles and books on analytical modeling (e.g., the deterministic influence diagrams of Howard and Matheson <1984>), artificial intelligence (e.g., the conceptual graphs of Sowa <1984>), database systems (e.g., the entity-relationship diagrams of Chen <1976>), software design (e.g., data flow diagrams as in Enos and Van Tilburg <1981> or Yourdon and Constantine <1979>), and even finance (e.g., the well known "DuPont graph" given, among other places, on p.229 of Weston and Copeland <1986>).

Extracts of genus graphs often suffice for explanatory purposes when the entire genus graph is overwhelming or unnecessary. Figure 19 shows an extract from the full genus graph for the capital expenditure planning model: namely, all nodes that reach one target node (note that genus names have been replaced by their associated key phrases). This diagram has been used to explain the capital feasibility test. The explanation proceeds from the top down until the curiosity of the questioner is satisfied. The dialog went something like this:

Q. How does the Capital Feasibility Test work?

A. It checks whether Total Capital Expenditures are within the Capital Limits.

Q. Total Capital Expenditures?

A. Yes, this is the simple sum of Portfolio Capital Expenditures and all Other Capital Expenditures outside of the portfolio.

Q. Oh, I see.

The modular tree can also be a useful communication graphic.

4. IMPLEMENTATION

Three prototype structured modeling systems are complete or nearly so. The first two, LEXICON (Clemence <1984>) and IIS (Farn <1985>), were designed early in the evolution of structured modeling and are discussed only briefly. The third, FW/SM, is now being developed at UCLA under the author's direction. All three adopt the non-graphical notational style (text-based schema cum elemental detail tables) described in Section 2.2. Other prototypes are at an earlier stage of development and will not be discussed here.

4.1 *Structured Modeling Systems*

An interactive structured modeling environment in the non-graphical style has four main components:

Model Library	schemata for a variety of models of interest to the organization, some perhaps with associated elemental detail
Solver Library	a collection of solvers to use in applications, typically including solvers for retrieval, simultaneous equations, and optimization
System Programs	code that creates the desired functionality of the structured modeling system
Workspace Store	workspaces in development or previously developed.

The *workspace* is the primary face of the system seen by the user. It is model-specific and consists of four parts:

Schema Section	here the user composes, maintains, and browses the text for a single schema
Elemental Detail Section	here the user loads, maintains, and browses the elemental detail tables associated with the schema
Reference Section	system-generated reference documentation corresponding to the schema and elemental detail; the user can manipulate all exhibits using suitable editors (see Section 4.3)
Activity Section	here the user does retrieval, opti-

mization, other needed model manipulations, and many other things throughout the modeling life-cycle.

The overall organization of the system should be hierarchical, and the user interface should be modern (e.g., Jarke and Vassiliou <1985>, Shneiderman <1987>) and should be reasonably appealing to users who are application domain oriented rather than modeling oriented.

Necessary functional capabilities include:

"Schema Compiler"	Read the schema, check the syntax and higher level structure, create skeletal elemental detail tables with resident expression evaluation capability, and configure a schema-directed loader/editor to facilitate data entry and maintenance of the elemental detail tables.
"Reference Section Maker"	Create all Reference Section exhibits automatically.
"Solver Interfaces"	Accept model manipulation requests expressed in terms of the schema and elemental detail tables and invoke the user-requested solver for retrieval, simultaneous equation solution, optimization, etc.
"Workbench Facilities"	Make available an outliner, text editor, table editor, graph maker, file and case manager, statistical analyzer, and other utilities that may be needed during the modeling life-cycle, especially in connection with the Activity Section. These facilities should be well-integrated and as interactive as is practical.

Other kinds of functional capabilities may be appropriate for special application contexts.

4.2 *Overview of All Three Implementations*

Figure 20 gives a brief summary of the three prototypes by functional capability.

LEXICON was done in FORTRAN on a mainframe and is aimed mainly at supporting optimization for large linear programming problems. It links to the advanced optimization system XS of

	LEXICON	IIS	FW/SM
SCHEMA COMPILER			
Structure ED Tables	Implicit	Implicit	Yes
Resident Evaluation	No	Yes	Yes
Smart Loader/Editor	No	Yes	Yes
REFERENCE EXHIBITS	Some	Several	Many
SOLVER INTERFACES			
Ad Hoc Query	No	Yes	Yes
Optimization	LP	LP (some)	Network, LP, NLP
WORKBENCH FACILITIES	Minimal	Some	Many

Fig. 20 Comparison of Three Prototype Implementations

Brown and Graves <1984>. The schema and data files are entered and processed in batch mode, but otherwise user interaction with LEXICON is menu-driven and dialog-based. There is no four-section "workspace". Only a subset of the full schema syntax is supported. Elemental detail tables are modified in form and not directly accessible to the user.

IIS was done in *KnowledgeMan* for the IBM PC/XT and is aimed mainly at demonstrating the feasibility of a hybrid information/analytical modeling system. It includes an interface to *LINDO* for a restricted class of linear programs. The workbench facilities available are those supplied by *KnowledgeMan*. The system is totally menu-driven; it has no "workspace", but a similar effect is achieved through its hierarchical menu structure. Only a subset of the full schema syntax is supported and elemental detail tables are modified in form and not directly accessible to the user.

FW/SM is being done in *Framework* for the IBM PC/AT and is aimed at creating a fully functional structured modeling system. *Framework* is a well known integrated multi-function program with built-in word processing, outlining, spreadsheet, flat file database, business graphics, and telecommunications. Whatever is not supported by these facilities can be programmed in *Framework*'s built-in language, or can be done by programs in any language by running them in a "DOS window" without leaving the *Framework* environment. The user interface is visually oriented and, in keeping with one of the recurring themes of structured modeling, tree-structured.

Flexible single table retrieval capability is available from native *Framework* facilities, and multi-table retrieval capability has been added. Two types of interfaces to optimization codes are used: control table and automatic. A control

table interface for generalized network optimization is operational, and two fully automatic interfaces for linear and nonlinear programming are under development.

Further details are given in the next subsection.

4.3 *The UCLA Implementation FW/SM*

This section reviews the main components of FW/SM according to the anatomical view presented in Section 4.1.

Model Library

Presently the Model Library contains about fifty schemata divided into six categories: administrative, database, economics, engineering, management science/operations research, and miscellaneous. Many are simple examples drawn from the major modeling paradigms, but others are translations of models used by various authors to illustrate alternative modeling frameworks or systems. The latter permit comparisons between structured modeling and these alternatives. A few examples:

- the static Mexican steel industry model from Kendrick, Meeraus, and Alatorre <1983> (GAMS)
- the demonstration problem from Ellison and Mitra <1982> and Lucas, Mitra, and Darby-Dowman <1983> (the UIMPS and CAMPS systems for LP)
- the educational database example from Chapter 27 of Date <1981> (relational, hierarchical, and network data models)
- the shipping industry example from Hammer and McLeod <1981> and the world traveler example from Hull and King <1986> (semantic database models).

Solver Library

Presently there are four solvers in the Solver Library that have been or are being interfaced:

- GENNET (Brown and McBride <1984>) for generalized network flow optimization
- *GINO* for nonlinear programming
- *LINDO* for linear and integer programming
- a query processor for a simple relation algebraic query language for ad hoc retrieval.

Some of the interfaces are discussed at the end of this section.

System Programs

System programs are written in FRED (the object-oriented language provided as an integral part of *Framework*), in PASCAL, or in C. FRED programs are executed interpretively, while compiled PASCAL and C programs are run through *Framework*'s DOS window. These programs are, of course, invisible to the user. The user sees only a menu of processing options.

Schema Section

The reader has already seen several examples of schemata (Figures 5, 7, and 10). In *Framework*, it is natural to make each genus paragraph its own "word frame" and each module paragraph a "containing frame." The two main features of the Schema Section are a syntax checker and *Framework*'s built-in tree-oriented editor.

The syntax checker is, like all processing options, invoked by simple menu selection. It issues diagnostics that pinpoint departures from the lexical, syntactic, and higher order structure prescribed for schemata.

The tree-oriented editor combines the capabilities of word processing with the capabilities of what is sometimes called an "outliner" (e.g., Dickinson <1986>). An outliner organizes and manipulates blocks of text (or other material) according to a user-supplied hierarchical (tree-based) structure. There are facilities to alter this structure (create, delete, copy, or move nodes; delete, copy, or move subtrees), to navigate within it (move toward or away from the root, move to a sibling), and to control its display (hide or unhide subtrees, hide or unhide node content).

These outliner facilities are very useful in the Schema Section because the paragraphs of a schema are arranged according to the modular outline (see Figures 4, 9, and 14).

Elemental Detail Section

The reader has already seen several examples of elemental detail tables (see Figures 6, 7, and 11). Recall that, for whatever schema is present in the Schema Section, the Schema Compiler needs to create empty elemental detail tables complete with resident expression evaluation capability. In *Framework*, it is natural for each table to be its own "database frame" with a FRED-coded "defining formula" for the generic rule associated with each column corresponding to a function or test genus.

The two main features in this section are *Framework*'s built-in table-oriented editor and a schema-directed loader/editor produced by the Schema Compiler.

The table-oriented editor understands the language of rows, columns, and data formats. Among its functions are data entry in any of three views, flexible copy and move operations, single table data extraction via general logical expressions, sorting, and a variety of column statistics. This editor is more than adequate for most keyboard-oriented input and editing tasks arising for the elemental detail tables.

Greater efficiency is possible with a schema-directed editor (not yet implemented). It has three primary functions, all of them based on information contained in the schema: automatic entry, error-trapping, optional table operations, and prompting.

"Automatic entry" fills out columns whose contents are determined by the content of other columns. Again using the feedmix model as an example, the first two columns of the ANALYSIS table can be filled in automatically with the Cartesian product of nutrients and materials after both of those columns have been filled out for the NUTR and MATERIAL tables (note that the index set statement of the ANALYSIS paragraph of Figure 5 stipulates a full Cartesian product).

"Error-trapping" rejects unacceptable entries based on information in the schema. For example, an attempt to enter a text string or a negative number in the minimum daily requirement column of the NUTR table would be resisted or flagged because the range statement of the MIN paragraph in the schema stipulates that only nonnegative numbers are allowed. Error-trapping can be on-line or done en masse when the user wishes.

"Optional table operations" are mostly manipulations that the system can perform to bring an elemental detail table into conformity with an index set statement. For example, such a statement can specify that a certain binary relation (like transitivity) must hold for two stub columns over a common domain; the system can help to enforce the desired relation (e.g., by taking the transitive closure).

"Prompting" is really a menu of options which the user currently has depending on what prior data entry or editing has taken place. With the feedmix model (see Figures 5 and 6), for example, the option to enter the minimum daily requirements would not appear until the list of nutrients has been entered.

Reference Section

There are many different summary reports that one might want to see in connection with a Schema Section, whether newly composed or called up from memory. The same is true, although to a lesser extent, of the Elementa_ Detail Section. Instead of trying to anticipate and produce all of these, a surprisingly successful strategy is adopted: produce a few basic displays containing the essentials, and let the user manipulate them

interactively to obtain customized reports. The manipulation can be by *Framework*'s word processor and outliner for text displays, by *Framework*'s table-oriented editor for tabular displays, or by a graphics-oriented editor (not available within *Framework*) for graphic displays.

Here are some examples of the basic displays produced by FW/SM, and of typical manipulations. They are of three types: textual, tabular, and graphical.

Reference Section: Text Displays

The main two text displays are the Natural Language Summary and the Modular Outline.

The Natural Language Summary was introduced in Section 3.3 and illustrated in Figure 15. Tailoring it to specific audiences, as in Figure 17, is facilitated by the outliner which, as mentioned earlier, supports subtree hiding (e.g., hide all of the genus paragraphs under &MATERIALS) as well as the usual word processing functions. Subtree hiding and unhiding also provide versatile tools for on-line briefings; start with everything hidden except the direct descendents of the root, and dynamically unhide and re-hide detail as appropriate.

The Modular Outline (e.g., Figures 4, 9, and 14) is automatically available as *Framework*'s "outline view" of the Schema Section. Subtree hiding/unhiding is again the most useful manipulation.

Reference Section: Tabular Displays

The main two tabular displays are the Adjacency/Reachability Matrix and the Genus/Module Summary. *Framework*'s word processor and table-oriented editor are useful for manipulating them.

The *Adjacency/Reachability Matrix* is illustrated in Figure 21. Each entry indicates the number of steps it takes to go from the row genus to the column genus in the genus graph (two or more steps print as "2"). The modular outline is used to determine both row and column order; by the no-forward-reference property, the matrix is necessarily upper triangular. Reading this table columnwise indicates which genera definitionally influence the column genus. Reading the table rowwise indicates which genera are definitionally influenced by the row genus. Thus, for example, if an error is discovered in one of the analysis coefficient values, one can tell at a glance which other genera could be affected.

The *Genus/Module Summary* is illustrated in Figure 22. The NAME column contains all genus and module names in modular outline order. The SEQ and PATH columns provide two reference numberings to permit the rows to be restored to their original

order after rearrangement and to facilitate finding paragraphs in the schema. The first numbering is obvious. The second one encodes, in a standard way, the position of each node in the modular outline (Figure 4). The TYPE column indicates which of the five possible types each genus is. The TABLE column gives the elemental detail table name corresponding to each genus. Finally, the KEY PHRASE column gives the (underlined) key phrase appearing in the interpretation part of each genus and module paragraph.

Consider now a few of the easy but useful manipulations of these two tables that *Framework* supports. Sorting on the GENUS column facilitates row access to the Adjacency/Reachability Matrix because then the row names are alphabetized. To produce a list of all genera that reach ANALYSIS, one may sort on the ANALYSIS column and block-copy the first part of the resulting GENUS column. Alternatively, one may extract the desired rows by executing the selection formula "ANALYSIS > 0". The logic of a selection formula can be arbitrarily complex, as logical "and", "or", "not", and other operators are available.

Sorting the Genus/Module Summary on NAME produces an alphabetized dictionary of genus and module names; on TYPE produces (contiguous) lists of the modules and of the genera of each of the five types; on TABLE produces an alphabetized list of tables and the genera associated with each; and on KEY PHRASE produces an alphabetized list of key phrases. Row extraction via logical expressions can also be useful. For example, it is easy to extract just the rows corresponding to attribute genera in the &MATERIALS module.

Reference Section: Graphical Displays

The two main graphical displays of interest are the Genus Graph (e.g., Figure 3) and the Modular Tree (e.g., Figure 4). Their full implementation awaits the acquisition of suitable graphics tools. A graphic display editor would also be desirable, so that the user can (a) reposition nodes or change their representation, and (b) pan and zoom to overcome the limited size and resolution of the monitor screen. In the meantime, a partial implementation allows any one node of the genus graph to be displayed centrally on screen along with all adjacent arcs and nodes. In the style of *Javelin*'s "Diagram view", the genus graph can be "walked" by changing the central node to one of its neighbors.

In addition to supporting on-screen graphic displays, it is also desirable to be able to drive a graphics printer and/or plotter so that larger, more detailed exhibits can be prepared.

Activity Section

The main facilities required by the Activity Section for MS/OR applications are those for retrieval, those for optimization, and general workbench tools. We discuss each in turn.

Here are three simple examples of queries that could arise in connection with the feedmix model:

(a) List the materials in decreasing order of unit cost.

(b) List the materials used in quantity greater than 1 pound per day per animal and with unit cost greater than 2 dollars per pound.

(c) List the materials with above average analysis in those nutrients for which the current mix fails the nutrition test.

The ability to answer such queries is important in order to gain full advantage of the information in the Elemental Detail Section, and in order to support a modeling application over its entire life-cycle. A distinction should be made between queries that involve but a single table and those that involve more than one. The first query above involves just one elemental detail table, while the other two involve two tables.

Framework provides flexible facilities for ad hoc query of single elemental detail tables. A simple relation algebraic query language has been implemented to permit multi-table ad hoc query. The five basic relation algebraic operations are supported, and also a few other important ones like the "natural join" (see, e.g., Section 5.2 of Ullman <1982>). An interface to a more advanced relational database package should be feasible because of the result mentioned in Section 2.4 under feature (g).

One style of optimization solver interface is the *control table*, a non-procedural device by which the user instructs the optimizer how to operate on the model at hand. Figure 23 gives an example of the GENNET control table filled in so as to make GENNET solve the transportation model of Figures 10 and 11. This table is independent of the content of the elemental detail tables so long as the schema does not change. (The column headings never change, but the row entries must be tailored to each schema, i.e., to each class of network flow models.) The syntax is straightforward. The last line can be interpreted as follows: generate arcs for all rows in table LINK, with the tail nodes taken from the PLANT column and the head nodes taken from the CUST column; use unit costs from the COST column of table LINK, infinite upper flow capacities, zero lower flow capacities, and no gains or losses on the flows (the last three entries could have been omitted, as they coincide with the default values).

Another style of optimizer interface is to make it almost fully automatic. The user would simply issue a command that identifies the objective function (a single function element), the variables (usually a list of variable attribute genera), and the constraints (usually a list of test genera). For example, the command to solve the transportation model might be:

> "Choose FLOW to minimize $ subject to
> T:SUP, T:DEM using GENNET."

Two such interfaces are under development for *GINO* and any LP optimizer that reads standard MPS input.

It is not clear at this time which of these two styles of interface is best in which circumstance. There are advantages to requiring the user to understand both the model and the solver at hand well enough to be able to fill out a simple control table linking the two. On the other hand, there are advantages to minimizing the work required of the user.

Whichever interface style is adopted, no computer programming skill is required of the user and it should be easy to switch optimizers if more than one apply to the model at hand.

If structured modeling systems should come into wide use, there would be incentives for optimizer developers to make their solvers available for inclusion in the Solver Library with suitable interface facilities. First, it would leave them free to concentrate on algorithmic matters without the distraction of having to build matrix generators, report writers, and user interfaces up to current desktop standards. Second, it would give access to realistic test problems. And third, it would supply a broader potential user base.

Other kinds of solvers besides those for retrieval or optimization would be desirable additions to the Solver Library. For example, a general equation-solver would be useful (e.g., Derman and Sheppard <1985>).

For workbench facilities we rely at present mainly on the integrated facilities of *Framework*: word processing, outlining, simple database, business graphics, spreadsheet, file management, and telecommunications. As an illustration, Figure 24 shows two graphs produced via standard menu options from the first elemental detail table in Figure 7. Desirable additional facilities include an input/output form editor (e.g., Prichard <1985>), data extraction tools (e.g., Belanger and Kintala <1985>), interactive data analysis, and statistical analysis.

5. OPPORTUNITIES FOR FURTHER WORK

Structured modeling provides many opportunities for further research, development, and cross-fertilization with established fields. Some of these opportunities are indicated here under four headings: the structured modeling framework, model design, implementation design, and model management systems.

Seven types of expertise are especially useful: discrete mathematics, analytical modeling, decision support system design, database management, high-level programming language design, software engineering, and artificial intelligence. The relevance of the first three is obvious. The relevance of the others derives from certain interdisciplinary parallels.

The parallels can be stated as follows. Designing a framework for analytical modeling is analogous to designing a data model (e.g., Tsichritzis and Lochovsky <1982>), designing a programming language (e.g., Shaw <1984>), and designing a framework for knowledge representation (e.g., Brachman and Levesque <1985>). All of these design activities are centrally concerned with representational frameworks. Moreover, designing a model schema within a framework for analytical modeling is analogous to designing a database schema within a given data model, designing a computer program within a given programming language, and designing a knowledge base within a given framework for knowledge representation.

It follows that the fields of database management, programming language design, software engineering, and artificial intelligence all have potentially important contributions to make to structured modeling. In fact, work in any of these fields can inform all of the others.

5.1 *Structured Modeling Framework*

The structured modeling framework itself can be studied in a theoretical way.

Scope and Comparative Studies

It would be useful to have a better understanding of the representational scope of structured modeling and its relationship to other frameworks, including those from related fields.

Ordinary mathematical programming models, graph and network models, and spreadsheet models are among those that always can be rendered as a structured model. What kinds of models cannot be so rendered? Are some types of recursive models intractable?

It turns out that any relational database can be rendered as a structured model. This is argued constructively in a forthcoming technical report and is proven theoretically by Farn <1985> using first order logic (more on this below). Farn also shows that the Entity-Relationship data model of Chen <1976> is subsumed by structured modeling. What about other data modeling frameworks? One that has been examined in detail is the well known and influential Semantic Data Model (SDM) of Hammer and McLeod <1981>. Most of the semantic features of SDM can be rendered straightforwardly in structured modeling, and virtually all of the remaining ones violate one or another tenet of structured modeling (usually avoidance of redundancy or the desirability of divorcing general structure from detailed data). Several other data models appear worthy of careful examination.

Functional programming languages (e.g., Glaser, Hankin and Till <1984>) bear a strong kinship to structured modeling. These declarative (non-procedural) languages are more problem-oriented than conventional computer programming languages, have a simpler mathematical basis, and are better suited to exploiting certain highly parallel computer architectures. How does their expressive power relate to that of structured modeling? It is intriguing to note that, from the functional programming viewpoint, the differences between "modeling" and "programming" largely disappear.

An area where modeling and programming have often been confused is discrete event simulation. It might seem that structured modeling is not applicable to this area because it does not allow the kind of procedural programming often used in the past to accomplish such simulations, but a research direction is suggested elsewhere (Geoffrion <1986b>) that may bring discrete event simulation within the reach of structured modeling.

Is structured modeling general enough to encompass any of the knowledge representation frameworks used in artificial intelligence (e.g., logic, production rules, semantic networks, or frames)? If so, then it should be possible to build hybrid systems that include access not only to solvers for retrieval and optimization, the mainstay model manipulations of structured modeling, but also to some types of inference engines. If not, then what additional syntactic/semantic extensions does structured modeling require in order to represent AI knowledge bases?

Two knowledge representation frameworks warrant special discussion: first order logic and semantic networks.

First Order Logic

First order logic (FOL for short -- see, e.g., Barr and Feigenbaum <1981>) is important for several reasons. First, it is one of the foundations on which AI was originally erected. Second, it probably is the best developed mathematically of all

knowledge representation frameworks. Third, and most pertinent for the present discussion, FOL provides a common ground on which many alternative modeling frameworks can be understood and compared.

Levesque and Brachman <1985> have used FOL to help understand what seems to be an inherent trade-off between the expressiveness of knowledge representation frameworks and their computational tractability. Reiter <1984> has expressed the relational data model in FOL and used this view to illuminate questions relating to query definition, incomplete information, integrity constraints, and extensions with greater semantic expressiveness. Li <1985> has recast the Entity-Relationship model and Semantic Data Model in FOL, and used this view to show that the latter subsumes the former. As mentioned earlier, Farn also used this approach.

A clear understanding of the relationship between structured modeling and FOL should yield insights into the expressive power of structured modeling, both alone and in relation to other modeling frameworks that may be recast in terms of FOL. It should also serve as a useful compass when contemplating future changes in the structured modeling framework and, possibly, as a gateway leading to the eventual incorporation of inference engines into structured modeling.

Chari <1985> is looking into this relationship and doing a PROLOG implementation.

Semantic Networks

The term "semantic network" actually covers a diversity of representational formalisms based on attributed graphs (e.g., Brachman and Levesque <1985>). It is said to be the most popular of all approaches to knowledge representation.

Our interest in semantic networks is that it appears to be the closest of all knowledge representation approaches to structured modeling, particularly in the rich development presented by Sowa <1984>. This book is the culmination of a long-term effort to unify the foundations of artificial intelligence in terms of "conceptual graphs". Many conceptual graphs can be represented as structured models and, conversely, a subset of all structured models can be represented as conceptual graphs. This relationship is explained in some detail in Geoffrion <1986b>.

Of particular interest is Sowa's demonstration of a two-way mapping between conceptual graphs and first order logic. This provides a way of attacking the agenda set forth in the previous topic. It could also lead to a kind of inference theory for structured modeling analogous to that available for FOL.

Also of interest is Sowa's proposed two-way mapping between conceptual graphs and natural language. It may be possible to develop an analogous mapping between structured modeling and natural language.

Conceptual Modeling

"Conceptual modeling" is a term coined to symbolize the need to cross-fertilize and harmonize common modeling issues arising in three previously independent fields: *data modeling* in database theory, *knowledge representation* in artificial intelligence, and *programming language abstractions* in high-level language design (Brodie et al <1984>). This requires raising modeling to a higher plane of abstraction and generality.

Analytical modeling as practiced in MS/OR is an important and conspicuous omission from the list. The goal of conceptual modeling should be to find the common abstract ground of all four fields. Since structured modeling already provides a formal framework for analytical modeling and, as has been mentioned, for other types of modeling as well, it would be appealing to study how structured modeling can contribute to both the original and expanded mission of conceptual modeling.

Successful work along these lines would have two primary benefits. First, it would produce a deeper and more general understanding of the modeling process so that it can be practiced more as a science and less as an art. Those who understand this more general theory of modeling would be armed with concepts and distinctions that sharpen their ability to organize the complexity of reality in formal ways. Second, it could produce a correspondence between each modeling framework and a master set of modeling abstractions, whether the framework is from data modeling, knowledge representation, programming language abstractions, or analytical modeling. One would then be in a position to determine the relative power of the various modeling frameworks, and to translate more easily among them. This could be a powerful approach to many of the comparative studies issues raised earlier, and would complement the first order logic approach mentioned in that context.

Extensions

Extensions of the existing structured modeling framework are possible. One attractive possibility would be to allow attribute elements to have values that are specified only probabilistically. This would facilitate some types of stochastic modeling and Monte Carlo simulation. Another possibility would be to allow an infinite number of elements. This would, for example, allow a genus to represent a countable infinity of time periods and thus permit modeling infinite time horizons explicitly rather than implicitly.

Other extensions and refinements of a less radical nature may also be of interest. For example, the syntax and semantics of generic calling sequences, range statements, index set statements, and generic rules could be refined to facilitate expressing details that are presently awkward or impossible to express. There is room for considerable variation among implementations of structured modeling, and in fact beyond a certain level of detail most syntax probably should be implementation-specific.

Designing extensions is an area calling for considerable discretion. The unbridled pursuit of representational power in a modeling framework can easily lead to excessive complexity, to a loss of previously available functionality, or to incompatibility with the desirable features listed in Section 1.2. Often it is wiser to let the user of a modeling system carry the burden of certain model details rather than to impose the burden on the system.

5.2 *Model Design*

Assuming one variant or another of a structured modeling framework, how should one go about designing a model -- particularly the generic and modular structure -- for a given practical application? It is always possible to design different structures that are more or less equivalent for any particular situation, but not all of these are equally useful. Some will have better properties than others. Principles are needed to help guide the practitioner.

The interdisciplinary parallels noted at the outset suggest that it is useful to look to neighboring fields for related ideas and results that can be adapted to the special needs of structured modeling.

"Normal Form" Theory

Recall that designing a relational database schema is analogous to designing a structured modeling schema. The theory of functional dependency and "normal forms" has been developed to avoid troublesome insertion, deletion, and update anomalies for relational databases (e.g., Chapter 7 of Ullman <1982>). Are there similar issues to be studied for structured modeling? Structured modeling appears to be relatively free of such anomalies, but it remains to establish this formally and to devise countermeasures for such cases as may exist. Farn <1985> was the first to examine this area.

Program and System Design Techniques

Recall that designing a computer program is analogous to designing a structured modeling schema. Many criteria have been proposed for what constitutes a "good" computer program, including these adapted from Yourdon and Constantine <1979>: clarity

of intent, execution efficiency, correctness, maintainability, modifiability, flexibility, and generality. Each of these has an obvious meaning in the analogous context of structured modeling.

Computer scientists and experienced implementors have long pursued an understanding of how computer program and system design influences these and other criteria. They have not hesitated to propose design techniques; among them are modular design, top-down design, structured design, Jackson's method, HOS, SADT, and others (e.g., Yourdon <1975> and Enos and Van Tilburg <1981>). If much of this work is pragmatic, stylistic, or otherwise subjective in character, this may be due to the inherent difficulty of the task and does not necessarily reflect adversely on the utility of this work. Some of these contributions can help inspire guidelines and techniques for designing good structured modeling schemata.

One of the classic contributions in this vein is "structured programming" (see Dahl, Dijkstra, and Hoare <1972> and Wirth <1971>). The spirit of structured programming is reflected in any schema with a well-considered modular structure, for genera then will obey a transparent hierarchical organization and there will be no forward references.

Knowledge Base Design Techniques

Recall that designing a knowledge base is analogous to designing a structured modeling schema. Are there knowledge base design techniques that can be adapted to structured modeling?

Is it possible to design an expert system that can construct a rough structured modeling schema for a new situation?

5.3 *Implementation Design*

Turning structured modeling ideas into good computer implementations raises many design challenges.

Language and Compiler Design

How can the context-free schema language used in this paper be improved? Would non context-free languages offer any advantages? Is a syntax-directed editor practical (e.g., Reps, Teitelbaum and Demers <1983>)? If so, this would significantly enlarge the pool of potential users because nearly any language is much harder to write than to read. What is the best design for the Schema Compiler? If a command language is needed to support some of the work carried out in the Activity Section of the workspace, what should its syntax be?

To consider just one topic in a little more detail, recall our intent to endow a structured modeling system with ad hoc query capabilities in the tradition of database management systems. We know that we can adopt virtually any relational database query language since, as explained earlier, elemental detail tables can be viewed as a relational database. But it should be possible to do better than that because a structured modeling schema has much more semantic content than a relational database schema. Thus an enticing topic is how to design a schema-directed query language and processor that is both simpler to use and more powerful than whatever is adopted as the point of departure from the realm of relational database systems. See Farn <1985> for an early contribution along these lines.

Data Structures

Data structure design for elemental detail becomes important when, as is often the case in medium to large-scale applications, the total number of elements is in the thousands or higher.

User Interface

Is a fully graphic, rather than text-oriented, interface based on the genus graph practical for the Schema Section of the workspace? The work of Jones <1985> seems particularly applicable here because it specifically addresses attributed graphs (which, of course, is the basic mathematical formalism of structured modeling). Jones' work and its relation to structured modeling are discussed at some length in Geoffrion <1986b>.

What is the best user interface for the schema-directed loader/editor in the Elemental Detail Section? It must balance prompting capabilities against flexibility of data entry.

Ease of use is a design objective of a structured modeling system because accessibility to problem domain experts (managers, policy makers, etc.) and user productivity are major goals. This suggests exploring how to exploit the availability of key phrases and explicit definitional linkages in a schema in order to achieve something approaching natural language dialog throughout the workspace. Natural language techniques from artificial intelligence could be useful. A promising approach along these lines would be to apply the work of Sowa <1984> mentioned earlier.

Another desirable feature would be the ability to select automatically the most appropriate solver depending on the particular query posed by the user and the mathematical nature of the model. This poses some deep questions of problem recognition and classification.

Solver Interface

Given a particular solver, how can it be installed in the Solver Library and interfaced once and for all with the rest of the system?

The interface can take one of several forms. One is to provide for a nonprocedural control table for the user to fill out whenever the solver is to be invoked. Another is to make the interface fully automatic by constructing a program that can read any compatible structured model and construct the necessary solver inputs therefrom. Both approaches are discussed in Section 4.3 in the context of solvers for optimization.

If theoretical work on reconciling structured modeling with the knowledge representation frameworks of artificial intelligence is successful, then some new implementation design issues arise: how to package AI solvers (for reasoning, question-answering, and other purposes) for installation in the Solver Library, and how best to interface with them. Any translation of a model representation that may need to occur should be totally transparent to the user.

Factorable Programming Technology for Derivatives

Expression evaluation is supposed to be a resident capability of a structured modeling system and so is not ordinarily thought of as requiring a special solver. However, a special solver may well be required if first and perhaps higher order derivatives are desired for function element values viewed as functions of prior attribute element values. McCormick <1983> has shown how to calculate such derivatives efficiently and exactly if the functions in question are represented in so-called "factorable form". Roughly speaking, this means that each function must be expressed as compositions of simple sums, products, and univariate transformations.

Lenard <1986> observed that, for many models, a structured modeling element graph supplies the better part of the required factorable representations if a little care is exercised when designing the model schema. Can this observation be implemented so as to achieve efficient computation of derivatives with minimum inconvenience to the modeler?

Program Integration Techniques

A structured modeling system incorporates many standard capabilities along with the novel ones. It should be much more efficient to assemble such a system from existing components than to build it from the ground up. This calls for program integration techniques.

A number of techniques are sketched in a forthcoming technical report. These and others can be found in the literature and in existing systems (e.g., Vo <1985>). Which techniques are most suitable?

Continuous Time Models

Many models with continuous time dynamics require attribute and function element values to be entire functions of time. Evaluation can then involve solving differential equations and taking integrals, and different kinds of solvers may then become necessary (e.g., for optimal control) by comparison with those used for static or discrete time models. This poses implementation design problems that have not yet been studied.

Data Flow Computers

It turns out that the element graph of a structured model is essentially equivalent to a machine-level program for a data flow computer, a kind of parallel processing architecture that overcomes some of the limitations of conventional von Neumann computers (e.g., Ackerman <1982>). This suggests that evaluation could be an extremely efficient process on such a computer. Implementation design for structured modeling on data flow computers is an attractive and untouched topic.

5.4 *Model Management Systems*

It has been recognized during the last decade that better computer-based systems are needed to support modeling in organizations where there are many models and many users. This situation raises important issues in the management of information resources. A variety of approaches to these issues can be found in a rapidly growing literature. See, for example, Dolk and Konsynski <1985> and Palmer <1984>. An extensive bibliography has been compiled by Blanning <1986>.

It can be argued that a structured modeling system of the type envisioned here provides the kernel of a model management system. The Model Library already provides for multiple models, the Solver Library already provides for multiple solvers, and an explicit design goal is to support the entire modeling lifecycle, which typically involves many people spanning different roles. One can create a structured model of the Model Library itself to categorize models by type, purpose, users, files needed, and so on. One can do a similar thing for the Solver Library, the System Programs, the Workspace Store, and for the community of users. Such tools can help support the essential managerial functions of model management. The first papers in this general vein are Dolk <1986a> and <1986c>.

Several new research topics are suggested by a structured modeling approach to model management. Three are selected for mention here.

Translators

There is a practical need to convert existing models, data, and associated materials to and from the lingua franca of structured modeling. It is not realistic to expect structured modeling to become the *only* language used. Cohabitation with other languages and systems is inevitable. Thus translators are needed for conventional mathematics and other modeling languages, data processing applications, and systems for information management and decision support.

Data Acquisition Techniques

Data acquisition is a topic of importance to organizations with multiple data sources on computer media. Can tools for data acquisition be designed so as to be schema-directed, that is, able to acquire data that are based on inferred or user-supplied correspondences between a model schema of interest and models describing the data sources?

Formalizing Model Schema Operations

It may be possible to formalize the basic operations over model schemata used for stepwise schema refinement, model comparison, model integration, and other kinds of development or model management work. For example, an important operation is joining two schemata together in such a way that equivalent genera are merged. Relatively few operation types probably account for most of the operations performed in practice. Formalization could bring orderly thought to many activities that would otherwise be ad hoc, and could lead to improved computer-based support for important classes of activities.

There are at least three possible approaches to formalization. First, take a tree manipulation approach based on modular structure. Second, take a graph grammar approach based on an attributed graph view of structured modeling (cf Jones <1985>). Third, take a formation rule approach based on a semantic network view of structured modeling (cf Sections 3.5 and 3.6 of Sowa <1984>). All three deserve exploration.

6. CONCLUSION

Structured modeling is a style intended to produce high quality model-based work with greater productivity and user acceptance. To achieve this objective it will be necessary to develop professional quality modeling environments based on these ideas and to produce cogent pedagogical materials for practitioners. These materials should also explain how to use structured modeling ideas in conjunction with conventional software.

The current prototype implementation FW/SM is a useful step toward professional quality software for structured modeling. It has helped to refine the original vision of a computer-based structured modeling environment and will continue to shed light on a variety of issues as new features are added and experience with it accumulates.

The development of pedagogical materials for practitioners is still in its early stages. Nevertheless, a few pioneers have already undertaken development work aimed at practical application.

It is too early to say whether systems based on structured modeling will succeed in providing the answer to some of the problems and opportunities facing MS/OR and kindred communities. Whatever the outcome, we submit that the eight design objectives of Section 1.2 merit serious attention by designers of new systems.

Serious attention is also merited by the striking interdisciplinary parallels pointed out in Section 5 between analytical modeling, database management, programming languages and software engineering, and artificial intelligence. Cross-fertilization is a most attractive undertaking. Progress in any of these fields informs the others.

The challenges of trying to conceive and bring into being a new generation of modeling systems are exciting and important. However, one should keep in mind that *language influences how people think*. Any coherent modeling system provides a "language" for modeling, and so must influence how its users think when modeling or doing model-based analysis. We saw in Section 3, for example, that a structured modeling system leads naturally to top-down and integrated approaches to model design, and to certain styles of communication with lay audiences. Are these influences truly beneficial? What other, perhaps less apparent, influences are there? We need to understand these issues in the broad context of rational approaches to mankind's organized activities, not only for structured modeling, but also for alternative modeling approaches. This is the true challenge of making modeling more of a science.

BIBLIOGRAPHY

ACKERMAN, W. <1982>. "Data Flow Languages," *Computer*, 15:2 (February), 15-25.

AHO, A.V., R. SETHI and J.D. ULLMAN <1986>. *Compilers*, Addison-Wesley, Reading, MA.

BARR, A. and E.A. FEIGENBAUM <1981>. *The Handbook of Artificial Intelligence*, Volume 1, William Kaufmann, Los Altos, CA.

BELANGER, D.C. and C.M.R. KINTALA <1985>. "Data Extraction Tools," *AT&T Technical J.*, 64:9 (November), 2025-2035.

BISSCHOP, J. and A. MEERAUS <1982>. "On the Development of a General Algebraic Modeling System in a Strategic Planning Environment," *Math. Programming Stud. 20* (October), North-Holland, Amsterdam, 1-29.

BLANNING, R.W. <1986>. "Tutorial on Model Management," paper presented at HICSS-19, Hawaii, January.

BODILY, S. <1986>. "Spreadsheet Modeling as a Stepping Stone," *Interfaces*, 16:5 (September-October), 34-52.

BRACHMAN, R.J. and H.J. LEVESQUE <1985>. *Readings in Knowledge Representation*, Morgan Kaufmann, Los Altos, CA.

BRODIE, M.L. <1984>. "On the Development of Data Models," in Brodie et al <1984>.

BRODIE, M., J. MYLOPOULOS and J. SCHMIDT <1984>. *On Conceptual Modeling*, Springer-Verlag, Berlin.

BROWN, G.G. and G.W. GRAVES <1984>. "XS Mathematical Programming System," Graduate School of Management, UCLA, personal communication.

BROWN, G.G. and R. McBRIDE <1984>. "Solving Generalized Networks," *Management Sci.*, 30:12 (December), 1497-1523.

BURGER, W.F. <1982>. "MLD: A Language and Data Base for Modeling," IBM Research Division, San Jose, Research Report RC 9639 (#42311), September 14.

CHARI, S. <1985>. "Knowledge Representation Using Structured Modeling," Research Proposal, Graduate School of Management, Computers and Information Systems, UCLA, December.

CHEN, P.P.S. <1976>. "The Entity-Relationship Model: Towards a Unified View of Data," *ACM Trans. Database Systems*, 1:1 (March), 9-36.

CHILDS, C. and C.R. MEACHAM <1985>. "ANALYTICOL - An Analytical Computing Environment," *AT&T Technical J.*, 64:9 (November), 1995-2007.

CLEMENCE, Jr., R.D. <1984>. *LEXICON: A Structured Modeling System for Optimization*, Master's Thesis, Naval Postgraduate School, Monterey, CA, June.

CRECINE, J.P. <1986>. "The Next Generation of Personal Computers," *Science*, 231:4741 (February 28), 935-943.

DAHL, O.J., E.W. DIJKSTRA and C.A.R. HOARE <1972>. *Structured Programming*, Academic Press, London.

DATE, C.J. <1981>. *An Introduction to Database Systems*, Volume 1, Third Edition, Addison-Wesley, Reading, MA.

DERFLER, Jr., F. <1986>. "Micro to Mainframe," *PC Magazine*, 5:9 (May 13), 116-124.

DERMAN, E. and E.G. SHEPPARD <1985>. "HEQS - A Hierarchical Equation Solver," *AT&T Technical J.*, 64:9 (November), 2061-2096.

DICKINSON, J. <1986>. "The Business of Words: Outliners," *PC Magazine*, 5:6 (March 25), 199-220.

DOLK, D.R. <1986a>. "Data as Models: An Approach to Implementing Model Management," *Decision Support Systems*, 2:1 (March), 73-80.

DOLK, D.R. <1986b>. "A Generalized Model Management System for Mathematical Programming," *ACM Trans. Math. Software*, 12:2 (June), 92-125.

DOLK, D.R. <1986c>. "Model Management and Structured Modeling: The Role of an Information Resource Dictionary System," Dept. of Administrative Sciences, Naval Postgraduate School, August.

DOLK, D.R. and B. KONSYNSKI <1985>. "Model Management in Organizations," *Information and Management*, 9:1 (August), 35-47.

ELLISON, E.F.D. and G. MITRA <1982>. "UIMP: User Interface for Mathematical Programming," *ACM Trans. Math. Software*, 8:3 (September), 229-255.

ENOS, J. and R. VAN TILBURG <1981>. "Software Design," *Computer*, 14:2 (February), 61-83.

FARN, C.K. <1985>. *An Integrated Information System Architecture Based on Structured Modeling*, Ph.D. Thesis, Graduate School of Management, UCLA.

FOURER, R. <1983>. "Modeling Languages Versus Matrix Generators for Linear Programming," *ACM Trans. Math. Software*, 9:2 (June), 143-183.

FOURER, R., D.M. GAY and B.W. KERNIGHAN <1987>. "AMPL: A Mathematical Programming Language," Computing Science Technical Report No. 133, AT&T Bell Laboratories, Murray Hill, NJ 07974, January.

GASS, S. <1987>. "Managing the Modeling Process," to appear in *European J. Oper. Res.* (May).

GASS, S.I., H.J. GREENBERG, K.L. HOFFMAN and R.W. LANGLEY <1986>. *Impacts of Microcomputers on Operations Research*, North-Holland, New York.

GEOFFRION, A.M. <1986a>. "Capital Portfolio Optimization: A Managerial Overview," *Proc. National Communications Forum*, 40:1, 6-10.

GEOFFRION, A.M. <1986b>. "Modeling Approaches and Systems Related to Structured Modeling," Working Paper No. 339, Graduate School of Management, UCLA, July.

GEOFFRION, A.M. <1986c>. "Integrated Modeling Systems," Working Paper No. 343, Graduate School of Management, UCLA, November. To appear in *Proceedings of the Conference on Integrated Modeling Systems* (held at the University of Texas, Austin, October 1986).

GLASER, H., C. HANKIN and D. TILL <1984>. *Principles of Functional Programming*, Prentice/Hall International, Englewood Cliffs, NJ.

HAMMER, M. and D. McLEOD <1981>. "Database Description with SDM: A Semantic Database Model," *ACM Trans. Database Systems*, 6:3 (September).

HARRISON, T.P. and C.A. DE KLUYVER <1984>. "MS/OR and the Forest Products Industry: New Directions," *Interfaces*, 14:5 (September/October), 1-7.

HOWARD, R.A. and J.E. MATHESON <1984>. "Influence Diagrams," in R.A. Howard and J.E. Matheson (eds), *The Principles and Applications of Decision Analysis*, Strategic Decisions Group, Menlo Park, CA.

HOROWITZ, E. <1984>. *Fundamentals of Programming Languages*, Second Edition, Computer Science Press, Rockville, MD.

HULL, R. and R. KING <1986>. "Semantic Database Modeling: Survey, Applications, and Research Issues," Technical Report 86-201, Computer Science Department, University of Southern California, April 1.

JARKE, M. and Y. VASSILIOU <1985>. "A Framework for Choosing a Database Query Language," *Comput. Surveys*, 17:3 (September), 313-340.

JONES, C.V. <1985>. *Graph-Based Models*, Ph.D. Thesis, Cornell University.

KATZ, S., L.J. RISMAN and M. RODEH <1980>. "A System for Constructing Linear Programming Models," *IBM Systems J.*, 19:4.

KENDRICK, D.A. and A. MEERAUS <1987>. *GAMS: An Introduction*, The World Bank, January. The Scientific Press, Palo Alto, to appear.

KENDRICK, D.A., A. MEERAUS and J. ALATORRE <1983>. *The Planning of Investment Programs in the Steel Industry*, The Johns Hopkins University Press, Baltimore, MD.

KRASNOFF, B. and J. DICKINSON <1986>. "Project Database II," *PC Magazine*, 5:12 (June 24), 106-227.

LENARD, M.L. <1986>. "Representing Models as Data," *J. Management Information Systems*, 2:4, 36-48.

LEVESQUE, H.J. and R.J. BRACHMAN <1985>. "A Fundamental Trade-Off in Knowledge Representation and Reasoning," in Brachman and Levesque <1985>.

LI, Y.P. <1985>. "On Data Modeling Through Logic," Research Paper, Graduate School of Management, UCLA, February 8.

LUCAS, C. and G. MITRA <1985>. "CAMPS: Preliminary User Manual," Department of Mathematics and Statistics, Brunel University, U.K., July.

LUCAS, C., G. MITRA and K. DARBY-DOWMAN <1983>. "Modeling of Mathematical Programs: An Analysis of Strategy and an Outline Description of a Computer Assisted System," Report TR/09/83, Department of Mathematics and Statistics, Brunel University, U.K., October.

MARKOWITZ, H.M. <1979>. "SIMSCRIPT," in J. Belzer, A.G. Holzman and A. Kent (eds.), *Encyclopedia of Computer Science and Technology*, Marcel Dekker, New York.

McCORMICK, G.P. <1983>. *Nonlinear Programming*, Wiley, New York.

MYLOPOULOS, J. and H.J. LEVESQUE <1984>. "An Overview of Knowledge Representation," in Brodie et al <1984>.

OREN, T.I., B.P. ZEIGLER and M.S. ELZAS <1984>. *Simulation and Model-Based Methodologies: An Integrative View*, NATO ASI Series, Springer-Verlag, Berlin.

PALMER, K. <1984>. *A Model Management Framework for Mathematical Programming*, Wiley, New York.

PLANE, D.R. <1986>. *Quantitative Tools for Decision Support Using IFPS*, Addison-Wesley, Reading, MA.

PRICHARD, Jr., R.M. <1985>. "FE - A Multi-Interface Form System," *AT&T Technical J.*, 64:9 (November), 2009-2023.

REITER, R. <1984>. "Towards a Logical Reconstruction of Relational Database Theory," in Brodie et al <1984>.

REPS, T., T. TEITELBAUM and A. DEMERS <1983>. "Incremental Context-Dependent Analysis for Language-Based Editors," *ACM Trans. Programming Languages and Systems*, 5:3 (July), 449-477.

ROY, A., L. LASDON and J. LORDEMAN <1986>. "Extending Planning Languages to Include Optimization Capabilities," *Management Sci.*, 32:3 (March), 360-373.

SCHITTKOWSKI, K. <1985>. "EMP: A Software System Supporting the Numerical Solution of Mathematical Programming Problems," Working Paper, Institut fur Informatik, Universitat Stuttgart.

SHAW, M. <1984>. "The Impact of Modeling and Abstraction Concerns on Modern Programming Languages," in Brodie et al <1984>.

SHNEIDERMAN, B. <1987>. *Designing the User Interface*, Addison-Wesley, Reading, MA.

SOWA, J.F. <1984>. *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley, Reading, MA.

TSICHRITZIS, D.C. and F.H. LOCHOVSKY <1982>. *Data Models*, Prentice-Hall, Englewood Cliffs, NJ.

ULLMAN, J.D. <1982>. *Principles of Database Systems*, Second Edition, Computer Science Press, Rockville, MD.

VO, K.-P. <1985>. "IFS - A Tool to Build Integrated, Interactive Application Software," *AT&T Technical J.*, 64:9 (November), 2097-2117.

WALKER, W.E. <1982>. "Models in the Policy Process: Past, Present and Future," *Interfaces*, 12:5 (October), 91-100.

WESTON, J.F. and T.E. COPELAND <1986>. *Managerial Finance*, Eighth Edition, The Dryden Press, Chicago, IL.

WIRTH, N. <1971>. "Program Development by Stepwise Refinement," *Comm. ACM*, 14:4 (April), 221-227.

WRIGHT, G. <1986>. "An Integrated Marketing Information System Based on Structured Modeling," paper presented at the Workshop on Structured Modeling, UCLA, August.

YOURDON, E. <1975>. *Techniques of Program Structure and Design*, Prentice-Hall, Englewood Cliffs, NJ.

YOURDON, E. and L.L. CONSTANTINE <1979>. *Structured Design*, Prentice-Hall, Englewood Cliffs, NJ.

COMMERCIAL SOFTWARE REFERENCES

Framework II. Ashton-Tate, 20101 Hamilton Ave., Torrance, CA 90502.

GINO. LINDO Systems Inc., P.O. Box 148231, Chicago, IL 60614.

Guru. Micro Data Base Systems Inc., P.O. Box 248, Lafayette, IN 47902.

IFPS. Execucom Systems Corp., 3410 Far West Blvd., Austin, TX 78731.

Javelin. Javelin Software Corporation, One Kendall Square, Building 200, Cambridge, MA 02139.

KnowledgeMan. Micro Data Base Systems Inc., P.O. Box 248, Lafayette, IN 47902.

LINDO. LINDO Systems Inc., P.O. Box 148231, Chicago, IL 60614.

PAM. Ketron, Inc., 151 S. Warner Rd., Wayne, PA 19807.

APPENDIX: SOME FORMALITIES

A forthcoming technical report presents the concepts of structured modeling in numerous formal definitions, proves basic theoretical results, and develops in detail the non-graphical (text and table based) notational conventions sketched in Section 2.2.

This appendix quotes selected formal definitions and propositions from this report. This serves : (a) to answer questions that may arise from the informal definitions given in Sections 2.1 and 2.2 of this paper, (b) to supply certain details not given in Section 2, and (c) to facilitate comparing structured modeling with alternative modeling approaches, many of which are described in Geoffrion <1986b>.

A **primitive entity** *element is undefined mathematically.*

A **compound entity** *element is a segmented tuple of primitive entity elements and/or other compound entity elements.*
(A "segmented tuple" is a finite nonempty ordered list whose components are partitioned in a contiguous way.)

An **attribute** *element is a segmented tuple of entity elements together with a unique* **value** *in some* **range**.

A **function** *element is a segmented tuple of elements together with a* **rule** *that associates a unique* **value** *in some* **range** *to this tuple -- more precisely, in the case of non-entity elements, to the value of these elements provided these values fall within a prescribed* **domain**.

A **test** *element is like a function element, except that it has a two-valued range (True,False).*

The segmented tuple portion of an element is called its **calling sequence**. *An element B is said to* **call** *another element A if A appears in B's calling sequence. A* **calling sequence segment** *has the obvious definition.*

A collection of elements is **closed** *if, for every element in the collection, all elements in the calling sequence of that element are also in the collection.*

A closed collection of elements is **acyclic** *if there is no sequence (E1,E2,...,En-1,E1) such that E1 calls E2, E2 calls E3, ..., En-1 calls En=E1, where n>2 and the elements of the sequence are not necessarily distinct.*

An **elemental structure** *is a nonempty, finite, closed, acyclic collection of elements.*

A **generic structure** *is defined on an elemental structure as a collection of partitions, one for each of the five types of elements. The resulting mutually exclusive and exhaustive element sets are called* **genera** *(plural of* **genus**).

A generic structure satisfies the **generic similarity** *property if the following is true for every genus other than primitive entity genera: every element in the genus has the same number of calling sequence segments and all calls in a given segment are to the same genus; moreover, each segment calls the same genus for every element. When this property holds, one can speak in the obvious sense of one genus "calling" another, and of a "genus' calling sequence".*

A **modular structure** *is defined on a generic structure as a rooted tree whose terminal nodes are in 1:1 correspondence with the genera. The non-terminal nodes are called* **modules**. *The* **default modular structure** *corresponds to the simplest possible such rooted tree, namely the one with only one module (the root).*

A **monotone ordering** *of a modular structure defined on a generic structure satisfying similarity is specified by an* **order** *for each sibling set. These orders are extended in the usual way to obtain a partial order over all nodes except the root whereby any two nodes can be compared so long as neither lies on the rootpath of the other. This partial order is* **monotone** *in that it is consistent with the partial order on the terminal nodes induced by calls among genera; that is, if genus B calls genus A and A and B are descendents of distinct sibling nodes #1 and #2 respectively (A=#1 and/or B=#2 permitted), then #1 comes "before" #2 in their sibling order.*

A **structured model** *is an elemental structure together with a generic structure satisfying similarity and a monotone-ordered modular structure.*

The **modular outline** *of a monotone-ordered modular structure is the indented list representation corresponding to the preorder traversal.*

The **element graph** *of an elemental structure is an attributed directed graph with a node for every element and an arc from element B to element A if element A calls element B. Every node has an attribute denoting its type (primitive entity, compound entity, attribute, function, or test). Every non-entity node has another attribute giving its value, every attribute node has another attribute giving its range, and every function and test node has an attribute giving its rule. Every arc has two attributes; the first identifies the calling sequence segment to which it corresponds, and the second identifies its position within the segment.*

The **genus graph** *of a generic structure satisfying similarity is a directed graph with a node for every genus and an arc for every segment of every genus (primitive entity genera excepted) directed from the genus being called to the calling genus.*

A **model schema** *is any prescribed class of structured models that satisfies isomorphism in this sense: given any two models in the class, their modules and genera can be placed in 1:1 correspondence in such a way that (a) adjacency is preserved in the modular structure trees, and (b) corresponding genera have the same number of calling sequence segments and call corresponding genera from each segment.*

The following propositions give some of the basic theoretical results associated with the above concepts, with a minimum of commentary.

> **Proposition.** *In an elemental structure with a generic structure satisfying similarity, no element calls another element in the same genus.*

> **Proposition.** *Genus graphs are always acyclic.*

A well known property of acyclic directed graphs is that their nodes can be classified uniquely into **ranks** such that nodes of rank r (r>1) have incoming arcs only from nodes of lower rank including at least one node of rank r-1.

Element and genus graphs can be ranked, for both are acyclic. The next result asserts that these rankings are consistent when viewed in terms of elements. One consequence of this fact is that *no partition of elements comprising generic structure may put together elements of different type or elemental rank*, if generic similarity is to hold.

> **Proposition.** *Consider an elemental structure together with a generic structure satisfying similarity. The rank of any element based on the element graph is identical to the rank of the element's genus based on the genus graph.*

The next result gives a key property of the modular outline.

> **Proposition.** *If genus B calls genus A in a structured model, then A comes before B in the modular outline.*

Consider an elemental structure, together with a generic structure satisfying similarity and a modular structure. It is natural to wonder about the existence of a monotone ordering and how to construct one, for without a monotone ordering there can be no structured model. The following result gives one of two known characterizations of when a monotone ordering exists.

The characterization as stated is theoretical, but the proof provides a simple and constructive method (that has been implemented) for determining monotone orderings when they exist.

> **Proposition** (excerpt). *Consider an elemental structure, together with a generic structure satisfying similarity and a modular structure. A monotone ordering exists if and only if the following condition holds: for every sibling set of the modular structure tree, there is no sibling sequence (S1,S2,...,Sn-1,S1) such that some genus descendent of S1 calls some genus descendent of S2, some genus descendent of S2 calls some genus descendent of S3, ..., some genus descendent of Sn-1 calls some genus descendent of Sn=S1, where n>2 and the siblings in the sequence are not necessarily distinct.*

A similar issue arises relative to the situation where no modular structure is given. It follows from the second and last propositions that, given an elemental structure together with a generic structure satisfying similarity, the default modular structure always has a monotone ordering.

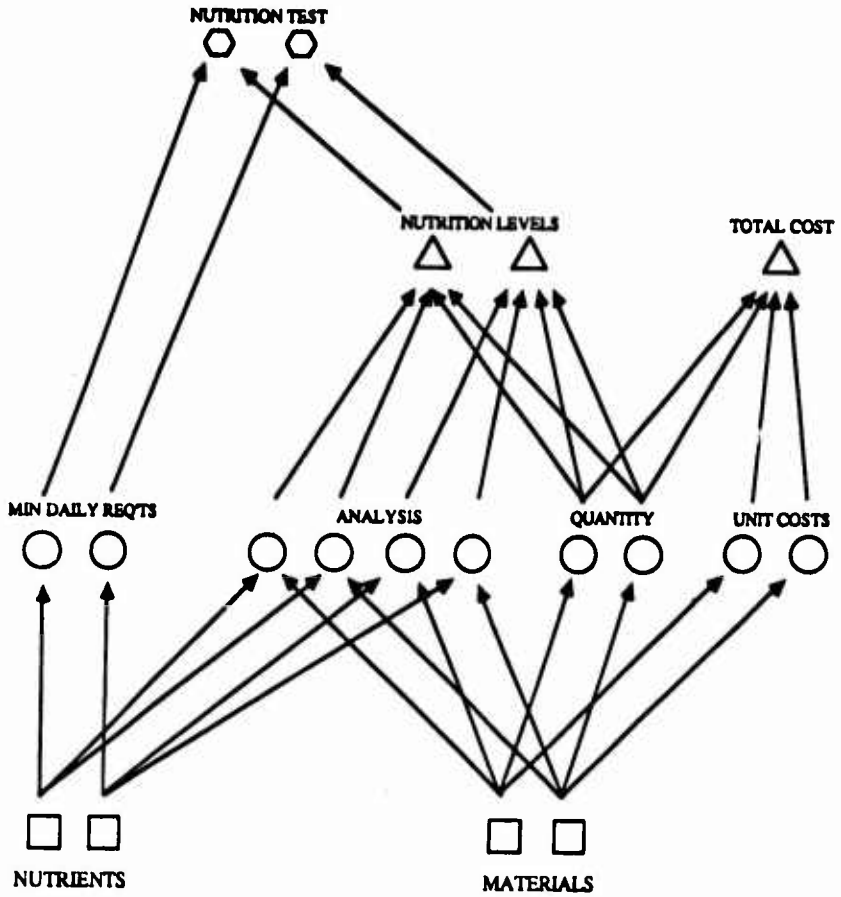


Fig. 1 Element Graph for a 2x2 Feedmix Model

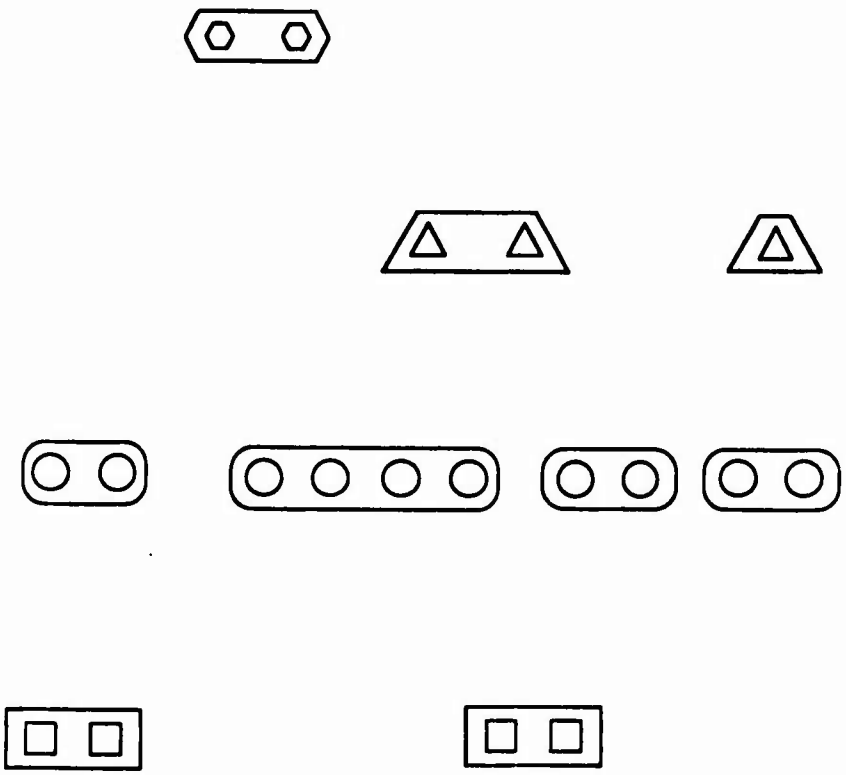


Fig. 2 Element Graph Nodes Partitioned by Type for Feedmix Model

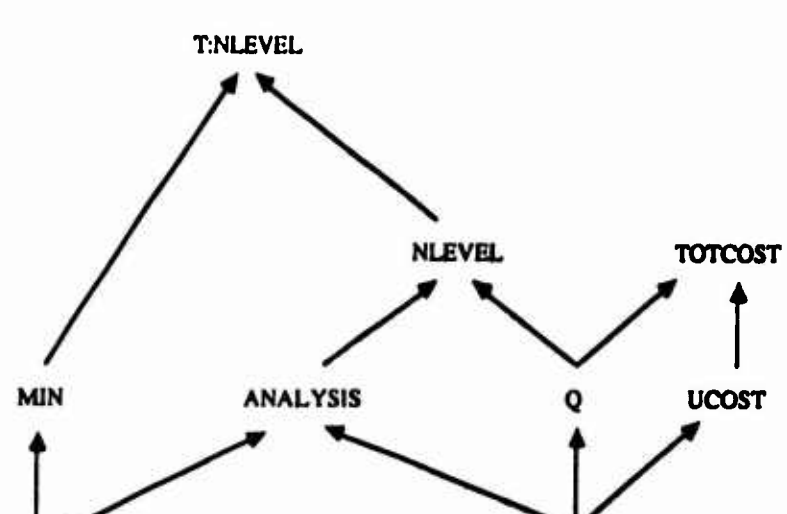


Fig. 3 Genus Graph for Feedmix Model

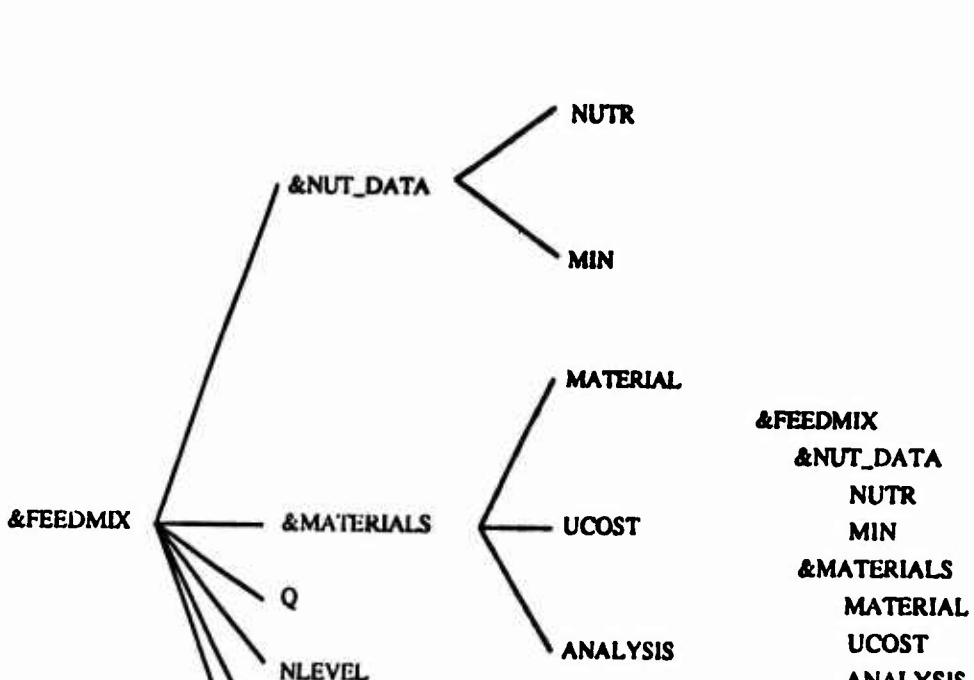


Fig. 4 Modular Tree and Modular Outline for Feedmix Model

&NUT_DATA *NUTRIENT DATA*

NUTRi /pe/ *There is a list of NUTRIENTS that animals require.*

MIN(NUTRi) /a/ (NUTR) : R+ *For each NUTRIENT there is a MINIMUM DAILY REQUIREMENT (units per day per animal) for the target animal population.*

&MATERIALS *MATERIALS DATA*

MATERIALm /pe/ *There is a list of MATERIALS that can be blended for animal feed.*

UCOST(MATERIALm) /a/ (MATERIAL) : R *Each MATERIAL has a UNIT COST ($ per pound of material).*

ANALYSIS(NUTRi,MATERIALm) /a/ (NUTR)x(MATERIAL) : R+ *Each NUTRIENT-MATERIAL combination has an ANALYSIS (units of nutrient per pound of material).*

Q(MATERIALm) /va/ (MATERIAL) : R+ *The feed QUANTITY (pounds per day per animal) of each MATERIAL is to be chosen.*

NLEVEL(ANALYSISi.,Q) /f/ (NUTR) ; SUMm (ANALYSISim * Qm) *Once the QUANTITIES are chosen, there is a NUTRITION LEVEL (units per day per animal) for each NUTRIENT calculable from the ANALYSIS.*

T:NLEVEL(NLEVELi,MINi) /t/ (NUTR) ; NLEVELi >= MINi *For each NUTRIENT there is a NUTRITION TEST to determine whether the NUTRITION LEVEL is at least as large as the MINIMUM DAILY REQUIREMENT.*

TOTCOST(UCOST,Q) /f/ ; SUMm (UCOSTm * Qm) *There is a TOTAL COST (dollars per day per animal) associated with the chosen QUANTITIES.*

Fig. 5 Schema for Feedmix Model

NUTR

NUTR	INTERP	MIN
P	Protein	16
C	Calcium	4

MATERIAL

MATERIAL	INTERP	UCOST
std	Standard Feed	1.20
add	Additive	3.00

ANALYSIS

NUTR	MATERIAL	ANALYSIS
P	std	4.00
P	add	14.00
C	std	2.00
C	add	1.00

Q

MATERIAL	Q
std	2.00
add	.50

NLEVEL

NUTR	NLEVEL	T:NLEVEL
P	15.00	FALSE
C	4.50	TRUE

TOTCOST

TOTCOST
3.90

Fig. 6 Sample Elemental Detail for Feedmix Model

ITEMi /pe/ *There is a list of ITEMS.*

&ITEMDATA *Certain ITEM DATA are provided.*

D(ITEMi) /a/ (ITEM) : R+ *Every ITEM has a DEMAND RATE (units per year).*

H(ITEMi) /a/ (ITEM) : R+ *Every ITEM has a HOLDING COST RATE (dollars per unit per year).*

F(ITEMi) /a/ (ITEM) : R+ *Every ITEM has a FIXED SETUP COST (dollars per setup).*

Q(ITEMi) /va/ (ITEM) : R+ *The ORDER QUANTITY (units per order) for each ITEM is to be chosen.*

&OPCON *OPERATING CONSEQUENCES following from ORDER QUANTITY choices.*

FREQ(Di,Qi) /f/ (ITEM) ; Di/Qi *Every ITEM has a SETUP FREQUENCY (average number of setups per year) equal to DEMAND RATE divided by ORDER QUANTITY.*

SETUP$(FREQi,Fi) /f/ (ITEM) ; FREQi * Fi *Every ITEM has an ANNUAL SETUP COST (dollars per year) equal to the SETUP FREQUENCY times the SETUP COST.*

CARRY$(Hi,Qi) /f/ (ITEM) ; Hi * Qi/2 *Every ITEM has an ANNUAL CARRYING COST (dollars per year) equal to its HOLDING COST RATE times one-half of its ORDER QUANTITY (which estimates average inventory level).*

ITEM$(SETUP$i,CARRY$i) /f/ (ITEM) ; SETUP$i + CARRY$i *Every ITEM has an ANNUAL ITEM COST (dollars per year) equal to its ANNUAL SETUP COST plus its ANNUAL CARRYING COST.*

TOT$(ITEM$) /f/ ; SUMi (ITEM$i) *The TOTAL ANNUAL COST (dollars per year) is the sum of all ANNUAL ITEM COSTS.*

ITEM

ITEM	D	H	F	Q	FREQ	SETUP$	CARRY$	ITEM$
COKE	3600	.40	9.00	500	7.20	64.80	100	164.80
7-UP	2500	.40	9.50	300	8.33	79.17	60	139.17
BEER	2000	.54	9.00	400	5	45	108	153

TOT$

TOT$
456.97

Fig. 7 Schema and Sample Elemental Detail for a Multi-Item EOQ Model

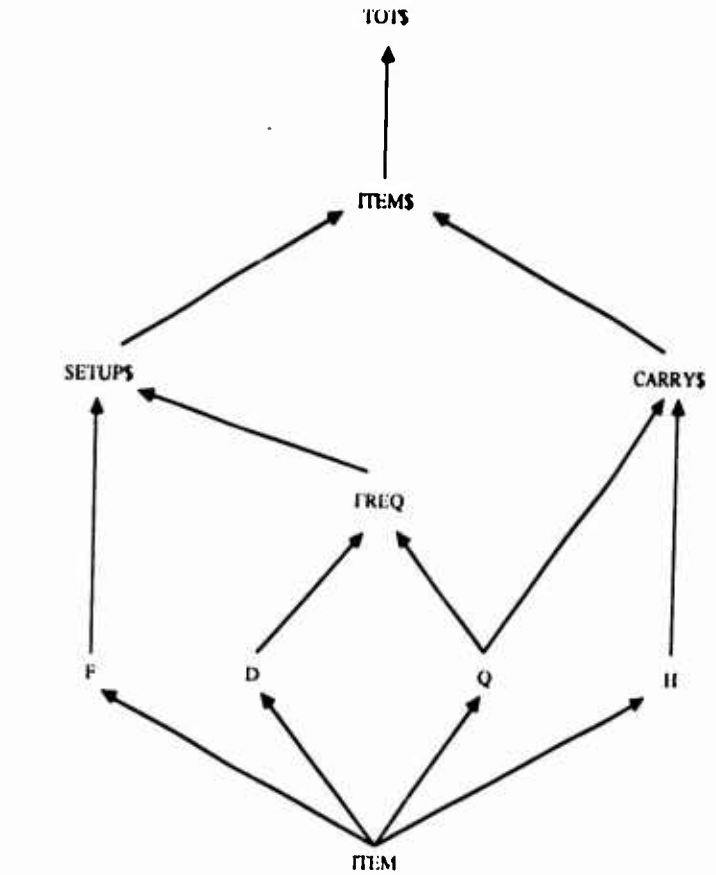


Fig. 8A Genus Graph for EOQ Model: First Pass

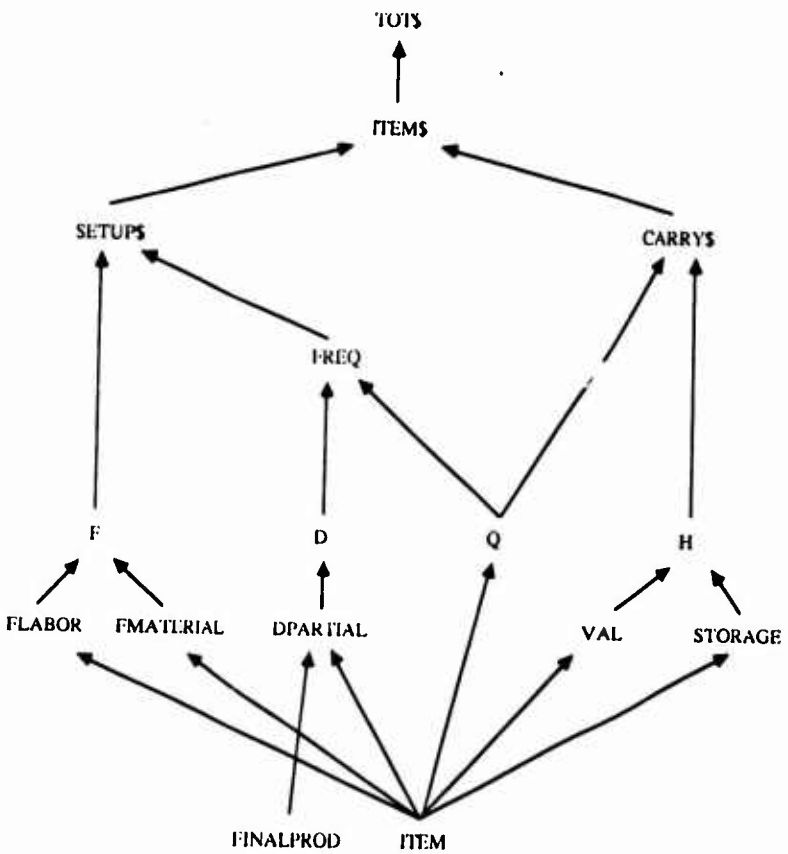


Fig. 8B Genus Graph for Multi-Item EOQ Model: Second Pass

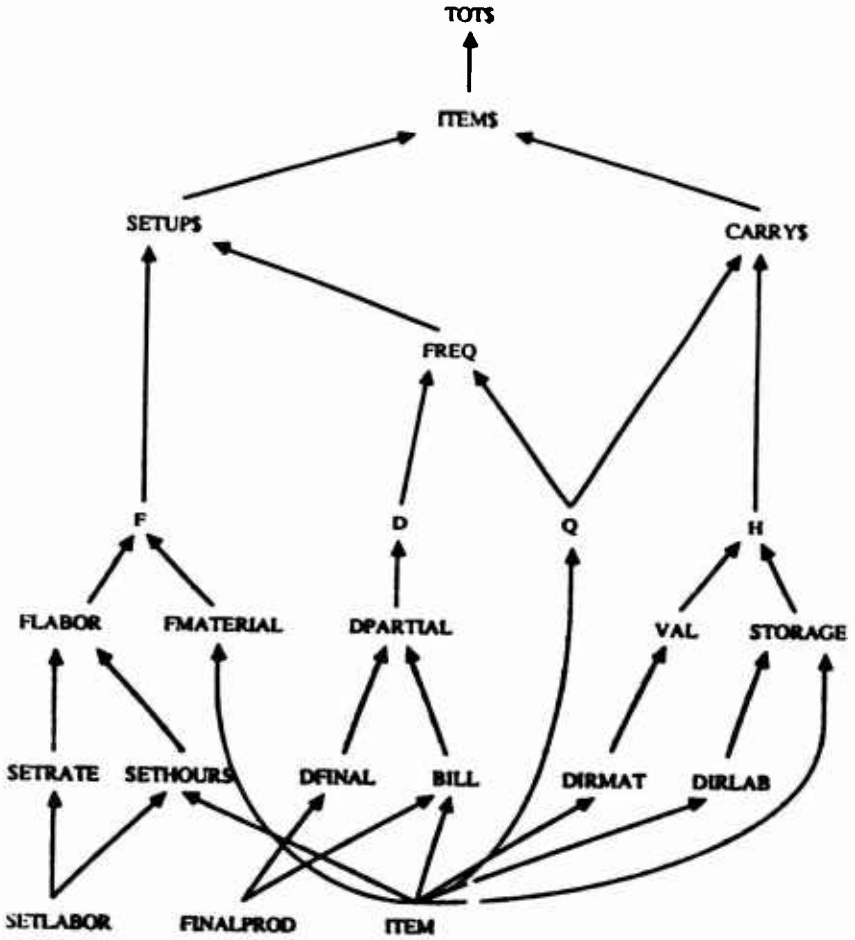


Fig. 8C Genus Graph for Multi-Item EOQ Model: Third Pass

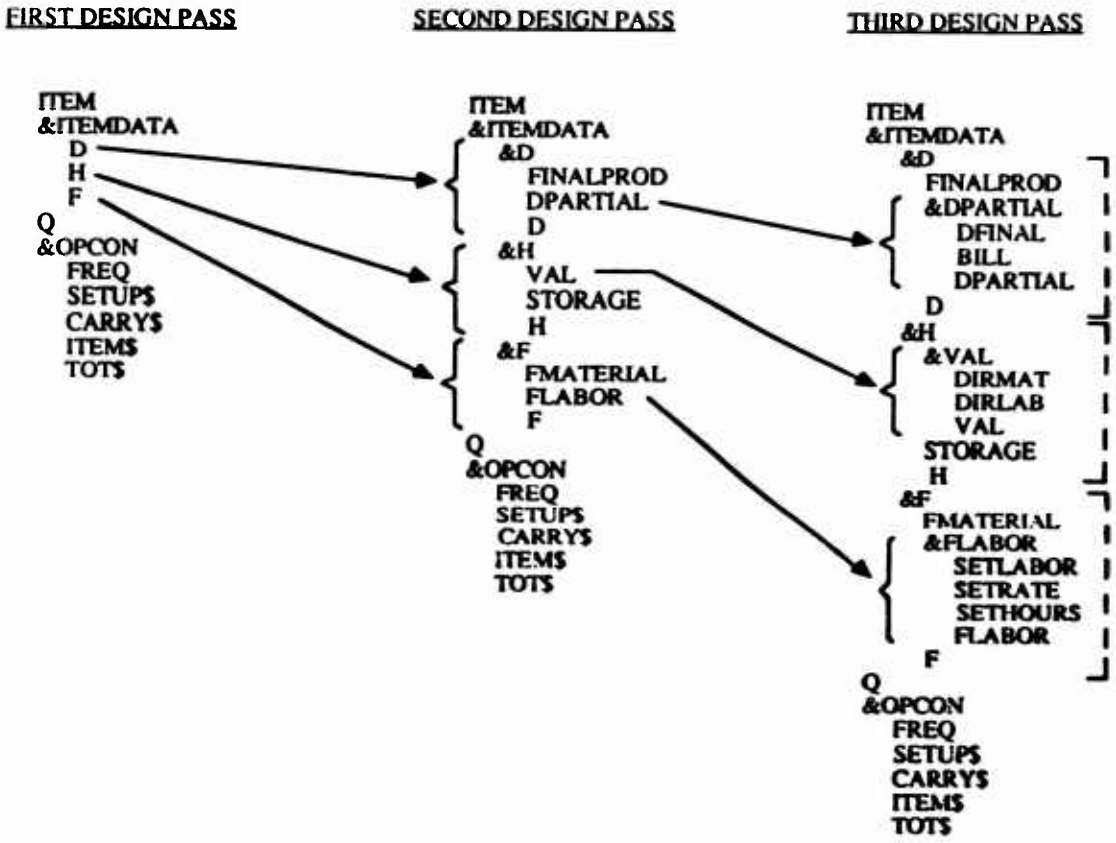


Fig. 9 Modular Outlines for Multi-Item EOQ Model: All Three Passes

```
&SDATA  SOURCE DATA

   PLANTi /pe/  There is a list of PLANTS.

   SUP(PLANTi) /a/ (PLANT) : R+  Every PLANT has a SUPPLY CAPACITY
   measured in tons.

&CDATA  CUSTOMER DATA

   CUSTj /pe/  There is a list of CUSTOMERS.

   DEM(CUSTj) /a/ (CUST) : R+  Every CUSTOMER has a nonnegative
   DEMAND measured in tons.

&TDATA  TRANSPORTATION DATA

   LINK(PLANTi,CUSTj) /ce/ Select (PLANT)x(CUST) where i covers
   (PLANT), j covers (CUST)  There are some transportation LINKS
   from PLANTS to CUSTOMERS. There must be at least one LINK
   incident to each PLANT, and at least one LINK incident to each
   CUSTOMER.

   FLOW(LINKij) /va/ (LINK) : R+  There can be a nonnegative
   transportation FLOW (in tons) over each LINK.

   COST(LINKij) /a/ (LINK) : R  Every LINK has a TRANSPORTATION COST
   RATE for use in $/ton.

$(COST,FLOW) /f/ : SUMi SUMj (COSTij * FLOWij)  There is a TOTAL
COST associated with all FLOWS.

T:SUP(FLOWi.,SUPi) /t/ (PLANT) : SUMj (FLOWij) <= SUPi  Is the
total FLOW leaving a PLANT less than or equal to its SUPPLY
CAPACITY? This is called the SUPPLY TEST.

T:DEM(FLOW.j,DEMj) /t/ (CUST) : SUMi (FLOWij) = DEMj  Is the total
FLOW arriving at a CUSTOMER exactly equal to its DEMAND?  This is
called the DEMAND TEST.
```

Fig. 10 Schema for Transportation Model

PLANT

PLANT	||	INTERP	SUP
DAL	||	Dallas	20,000
CHI	||	Chicago	42,000

CUST

CUST	||	INTERP	DEM
PITTS	||	Pittsburgh	25,000
ATL	||	Atlanta	15,000
CLEV	||	Cleveland	22,000

LINK

PLANT	CUST	||	FLOW	COST
DAL	PITTS	||	5,000	23.50
DAL	ATL	||	15,000	17.75
DAL	CLEV	||	0	32.45
CHI	PITTS	||	20,000	7.60
CHI	CLEV	||	22,000	25.75

$

||	$
||	1,102,250

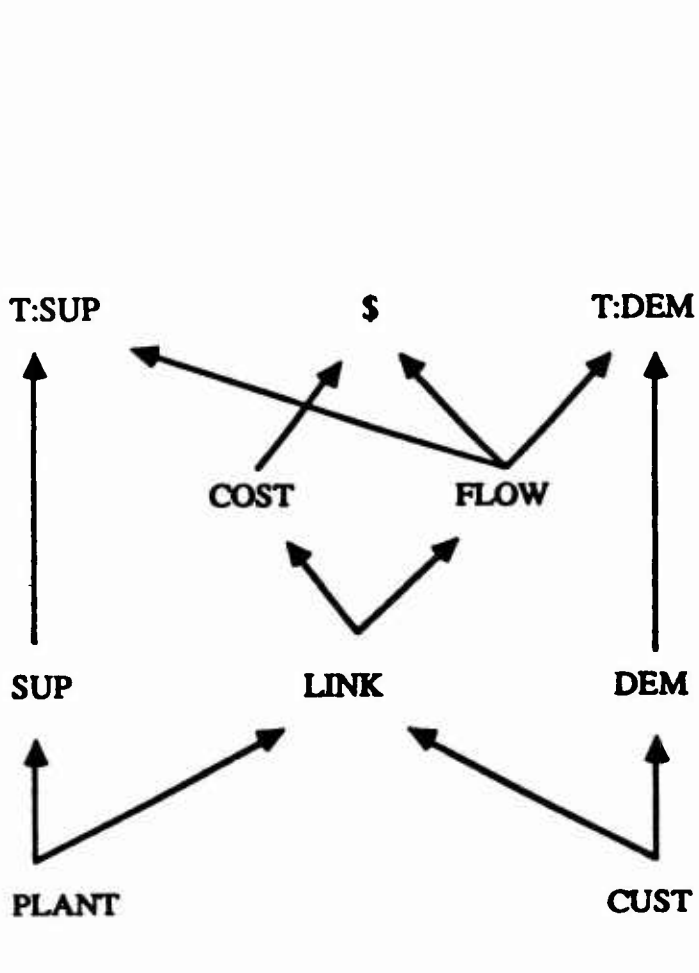
T:SUP

PLANT	||	T:SUP
DAL	||	TRUE
CHI	||	TRUE

T:DEM

CUST	||	T:DEM
PITTS	||	TRUE
ATL	||	TRUE
CLEV	||	TRUE

Fig. 11 Sample Elemental Detail for Transportation Model



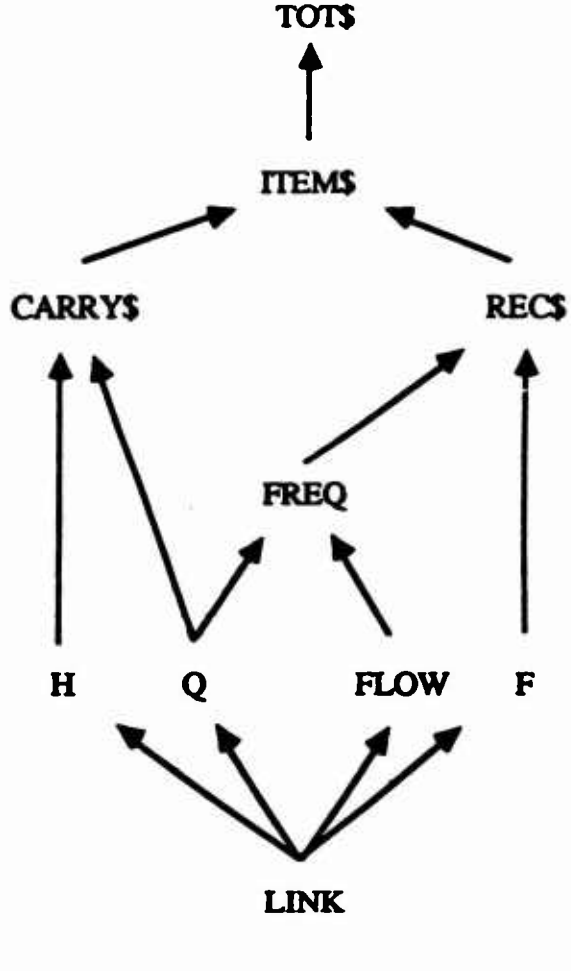


Fig. 12 Genus Graphs for Transportation and Modified EOQ Models

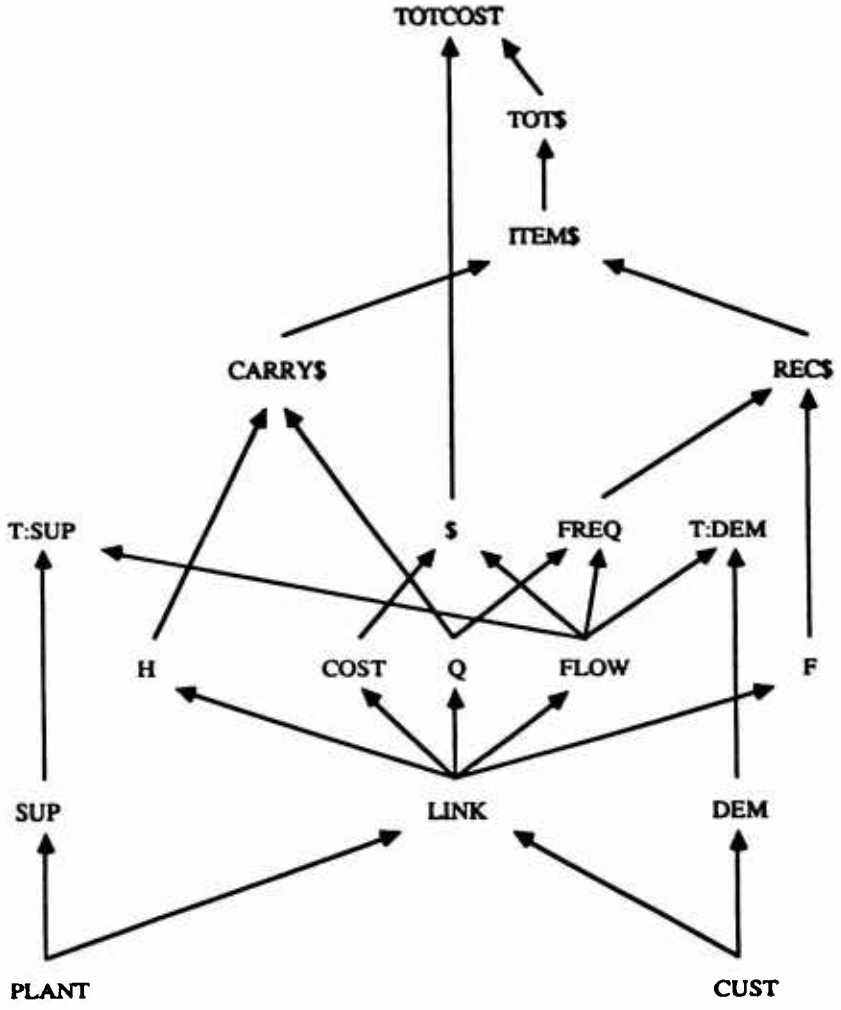


Fig. 13 Genus Graph for Integrated Model

```
&TRANS
    &SDATA
        PLANT
        SUP
    &CDATA
        CUST
        DEM
    &TDATA
        LINK
        FLOW
        COST
    $
    T:SUP
    T:DEM
&EOQ
    &ITEMDATA
        H
        F
    Q
    &OPCON
        FREQ
        REC$
        CARRY$
        ITEM$
        TOT$
TOTCOST
```

Fig. 14 Modular Outline for Integrated Model

&NUT_DATA NUTRIENT DATA

NUTRi There is a list of NUTRIENTS that animals require.

MINi For each NUTRIENT there is a MINIMUM DAILY REQUIREMENT (units per day per animal) for the target animal population.

&MATERIALS MATERIALS DATA

MATERIALm There is a list of MATERIALS that can be blended for animal feed.

UCOSTm Each MATERIAL has a UNIT COST ($ per pound of material).

ANALYSISim Each NUTRIENT-MATERIAL combination has an ANALYSIS (units of nutrient per pound of material).

Qm The feed QUANTITY (pounds per day per animal) of each MATERIAL is to be chosen.

NLEVELi Once the QUANTITIES are chosen, there is a NUTRITION LEVEL (units per day per animal) for each NUTRIENT calculable from the ANALYSIS.

T:NLEVELi For each NUTRIENT there is a NUTRITION TEST to determine whether the NUTRITION LEVEL is at least as large as the MINIMUM DAILY REQUIREMENT.

TOTCOST There is a TOTAL COST (dollars per day per animal) associated with the chosen QUANTITIES.

Fig. 15 Natural Language Summary for Feedmix Model

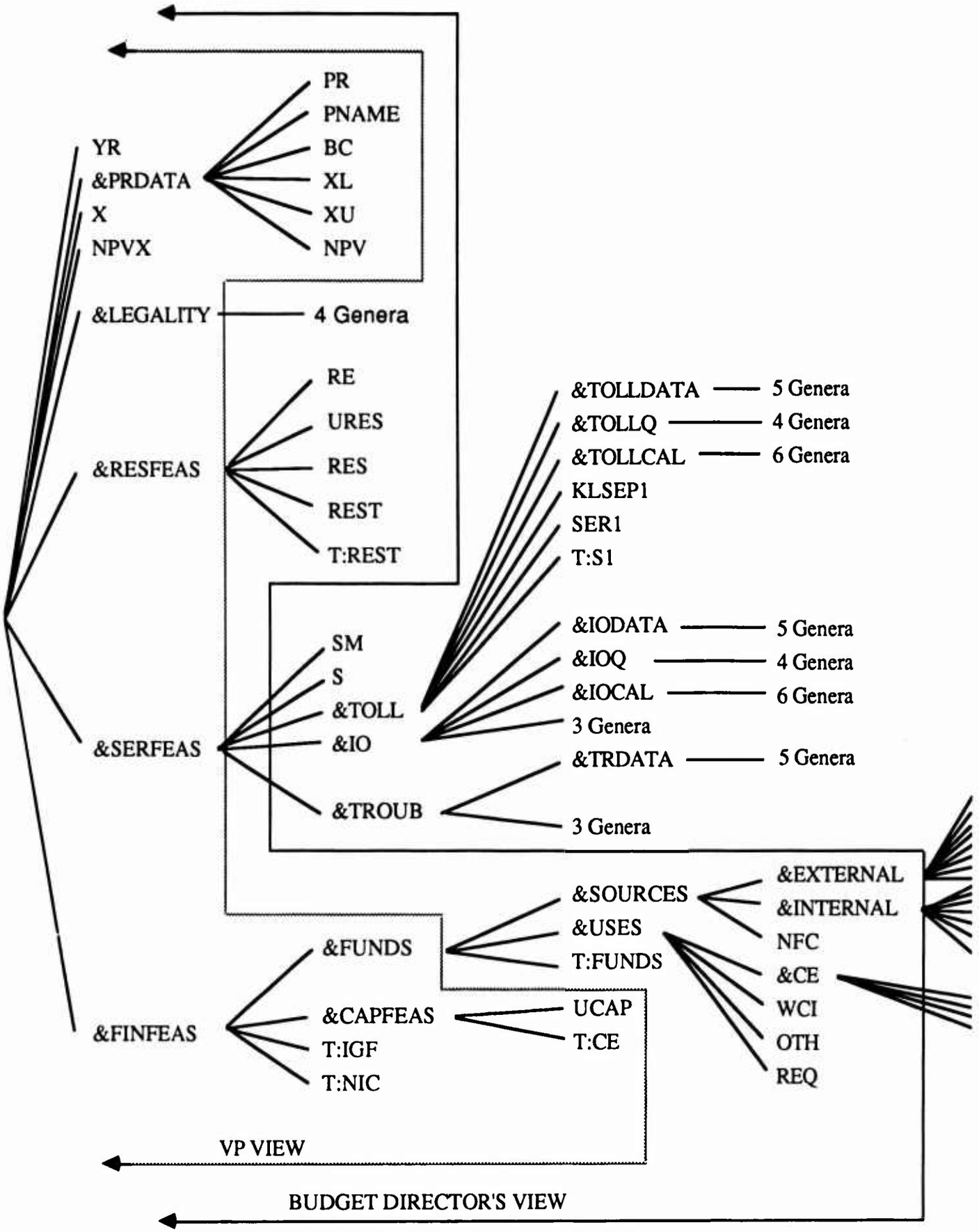


Fig. 16 Modular Tree for Capital Planning Model, with Two Views Indicated

YRt The model addresses a PLANNING HORIZON of five individual YEARS.

&PRDATA There are some BASIC PROJECT DATA.

PRp There is a list of candidate PROJECTS.

PNAMEp Each PROJECT has an EXTENDED PROJECT NAME.

BCp A DIVISIBILITY CODE, either "B" or "C", is assigned to each PROJECT to indicate whether it is indivisible (binary) or continuously divisible in character.

XLp A LOWER ACCEPTANCE LIMIT (a fraction) is specified for each PROJECT (the default is 0).

XUp An UPPER ACCEPTANCE LIMIT (a fraction) is specified for each PROJECT (the default is 1).

NPVp A NET PRESENT VALUE (NPV) is given for each PROJECT.

Xp An ACCEPTANCE LEVEL between 0 and 1 is to be chosen for each PROJECT; a complete set of choices defines a trial PORTFOLIO.

NPVX PORTFOLIO NPV is the primary index of a PORTFOLIO's merit. It is the sum over PROJECTS of ACCEPTANCE LEVEL times PROJECT NPV.

&LEGALITY The PORTFOLIO must be "LEGAL".

&RESFEAS RESOURCE FEASIBILITY is a desirable PORTFOLIO property.

&SERFEAS SERVICE FEASIBILITY is a desirable PORTFOLIO property.

&FINFEAS FINANCIAL FEASIBILITY is a desirable PORTFOLIO property.

&FUNDS The model incorporates a FUNDS STATEMENT based on the standard financial statement by the same name. It depends on the PORTFOLIO, is calculated for each YEAR, and plays a key role in defining FINANCIAL FEASIBILITY.

&CAPFEAS CAPITAL FEASIBILITY is an aspect of FINANCIAL FEASIBILITY.

UCAPt An UPPER CAPITAL LIMIT is given for each YEAR for the company as a whole.

T:CEt Given a trial PORTFOLIO, a CAPITAL FEASIBILITY TEST checks for each YEAR whether TOTAL CAPITAL EXPENDITURES are within the UPPER CAPITAL LIMIT.

T:IGFt Given a trial PORTFOLIO, an IGF FEASIBILITY TEST is applied to the FUNDS STATEMENT each YEAR to check whether the ratio of NET FUNDS FROM INTERNAL SOURCES to CAPITAL REQUIREMENTS is at least as large as a threshold value supplied by management.

T:NICt A NET INCOME FEASIBILITY TEST applies each YEAR to check whether NET INCOME TO COMMON is at least as large as a threshold value supplied by management.

Fig. 17 Natural Language Summary for VP View of Capital Planning Model

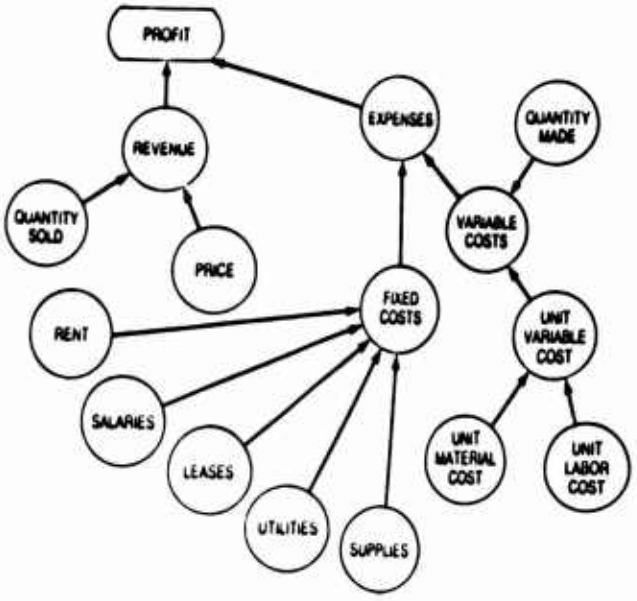


Fig. 18 Graphic from Plane <1986>

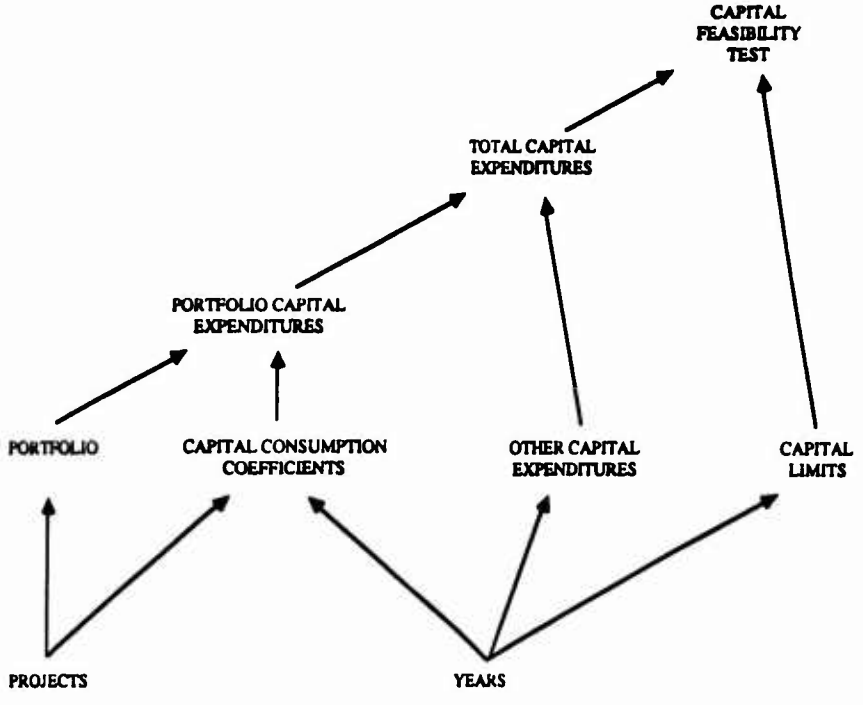


Fig. 19 Genus Graph Extract for Capital Planning Model

GENUS	NUTR	MIN	MATERIAL	UCOST	ANALYSIS	Q	NLEVEL	T:NLEVEL	TOTCOST
NUTR	0	1			1		2	2	
MIN		0						1	
MATERIAL			0	1	1	1	2	2	2
UCOST				0					1
ANALYSIS					0		1	2	
Q						0	1	2	1
NLEVEL							0	1	
T:NLEVEL								0	
TOTCOST									0

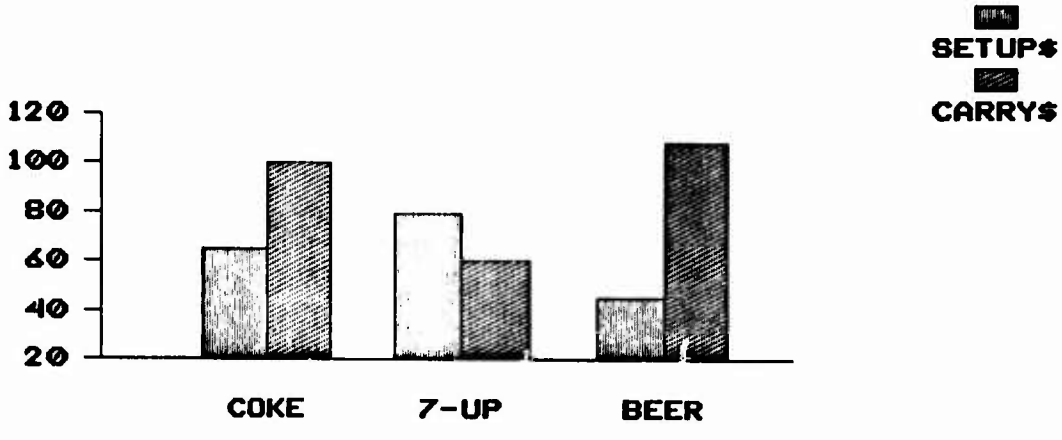
Fig. 21 Adjacency/Reachability Matrix for Feedmix Model

NAME	SEQ	PATH	TYPE	TABLE	KEY PHRASE
&NUT_DATA	1	1			NUTRIENT DATA
NUTR	2	1.1	PE	NUTR	NUTRIENTS
MIN	3	1.2	A	NUTR	MINIMUM DAILY REQUIREMENT
&MATERIALS	4	2			MATERIALS DATA
MATERIAL	5	2.1	PE	MATERIAL	MATERIALS
UCOST	6	2.2	A	MATERIAL	UNIT COST
ANALYSIS	7	2.3	A	ANALYSIS	ANALYSIS
Q	8	3	VA	Q	QUANTITY
NLEVEL	9	4	F	NLEVEL	NUTRITION LEVEL
T:NLEVEL	10	5	T	NLEVEL	NUTRITION TEST
TOTCOST	11	6	F	TOTCOST	TOTAL COST

Fig. 22 Genus/Module Summary for Feedmix Model

TAIL NODE	HEAD NODE	UNIT COST	UPPER CAP	LOWER CAP	MULT
PLANT.PLANT	PLANT.PLANT		PLANT.SUP		-1
CUST.CUST	CUST.CUST		CUST.DEM	CUST.DEM	1
LINK.PLANT	LINK.CUST	LINK.COST	INF	0	1

Fig. 23 GENNET Control Table for Transportation Model



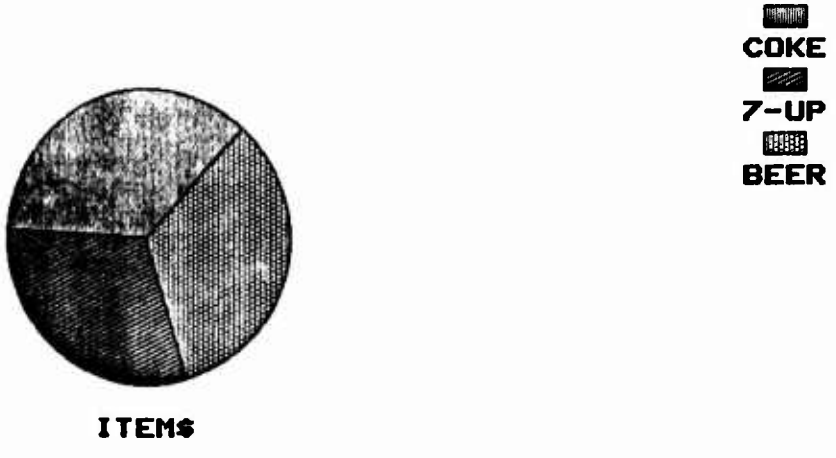


Fig. 24 Two Business Graphs for Multi-Item EOQ Model